B9

(12) UK Patent Application (19) GB (11) 2 168 831 A

(43) Application published 25 Jun 1986

(21) Application No 8428608	(51) INT CL⁴ H04L 11/26 G06F 12/14
(22) Date of filing 13 Nov 1984	(52) Domestic classification (Edition H): G4A AP
(71) Applicant Steebek Systems Ltd. (United Kingdom), 3 The Paddock, Hambridge Road, Newbury, Berkshire RG14 5TQ	(56) Documents cited GB 1588147 EP A1 0067611 EP A1 0100260 WO A1 83/02343
(72) Inventor David Robert Llewellyn Jones	(58) Field of search G4A
(74) Agent and/or Address for Service R. G. C. Jenkins & Co., 12-15 Fetter Lane, London EC4A 1PL	

(54) Password-protected data link

(57) A password-protected data communication system for transfer of data between remote user terminals and a host computer via public telephone lines and the like is further made secure by virtue of the fact that password transactions and/or interchanges are automatically effected between special modems provided at the user terminals and at the host computer without action or intervention (other than call initiating action) by the users who are denied access to or control over the passwords. A callback facility may also be provided and can be structured to enable users to communicate with the host from non-static locations.

GB 2 168 831 A

SPECIFICATION

Improvements relating to computer systems

5 This invention concerns improvements relating to computer systems and more particularly concerns the protection of host computers from unauthorised access via remote terminals coupled with the host computer over public communications net-
10 works including the public telephone system.

As is well known, it is customary to provide a user wishing to access a host computer, for example a database to be interrogated or searched by the user, with a unique user identification or pass-
15 word and to provide at the host computer a table of user identifications for which access to the computer database is permitted. The user's terminal is customarily connected via a modem to the public telephone network, for example, which in turn con-
20 nects via a corresponding modem with the host computer. The user, when wishing to access the host computer, calls the telephone number of the host computer, receives an answering tone when the telephone line connection is established, and
25 then enters his user identification via his terminal keyboard; the user identification must be received and verified at the host computer in order for access to be provided.

Whilst the provision of user identification pass-
30 words to be verified at the host computer before access is permitted does provide a baseline level of security against unauthorised access, nonetheless it does not in many situations provide for sufficient security. Computer systems which can be
35 reached through the public telephone system are potentially vulnerable to unauthorised acces by anyone who has by whatever means improperly come into possession of an authorised user identification password and further sophisticated com-
40 puter based techniques exist whereby unauthroised entry can be obtained once the dial-up telephone number of a computer facility has been obtained.

To further protect against such fraudulent ac-
45 cess, efforts have been made to implement less readily determinable user passwords, and also automatic disconnection of the incoming terminal line has been utilised following a small number of invalid attempts to enter an acceptable password.
50 A more recent proposal has been to provide a so-called port protection device external to the host computer's dial-up access ports, the port protection device having on-board microprocessor intelligence which is used to provide a level of external
55 password protection to any communication line.

The port protection device requires a potential dial-up terminal user to manually enter a password as a first step towards connecting with the host computer, and the device then compares this password
60 with a table of valid user passwords stored in its own memory. Only if the user-entered password matches a previously stored password in the port protection device memory is the user enabled to proceed with the routine logging-on procedure at
65 the host computer involving entry of a further

password etc.

As yet a further proposal, it has also been suggested to introduce a callback facility into a port protection device; since most legitimate users of a
70 host computer system can be presumed to have a routine work station at a fixed location, the rationale behind the callback proposal is that the port protection device would instruct a user to hang-up once his password had been verified and then
75 would call up a telephone number called from its own memory and associated in the memory with the password entered by the user; by this means only a user in possession of a proper password and located at the work station customarily associ-
80 ated with that password would be able to access the host computer.

According to the principal aspect of the present invention, it is proposed that the modems provided at each end of the data communication line, that is
85 at the user's terminal end and at the host computer end, automatically carry out the password transaction(s) or interchange(s) without action or intervention by the user who, in accordance with the invention, is denied access to or control over
90 the password(s). By this means, a very long and potentially indeterminable password comprising virtually an infinite number of possible character combinations (that is to say a virtually infinite "keyspace" size) can be utilised; by automatic use
95 of such a comprehensive password, which has many many more digits than could possibly be remembered and manually entered at a terminal, and by not revealing the password to the terminal end user much greater security of access is in-
100 sured.

In a practical situation therefore, the conventional modems which would customarily be provided at each end of the communication line would be replaced by special modems configured, in ac-
105 cordance with the invention, to include means for exchanging the necessary password(s), and means to enable password(s) to be entered during manufacture of the modem and, if desired, to the customer's specification, such means including, for
110 example, provision in the modem of appropriately programmed memory media. Autodial facilities would also be associated with each of the modems or at least with the user end modem.

In operation of a system in accordance with the
115 invention, the user will by appropriate operation of his terminal cause his modem to initiate a call to the host's modem, which requires the user's modem to transmit its preprogrammed password. On receipt of a valid password verified by comparison
120 with a password store at the host modem, the host's modem authorises direct connection of the user to the host system. Should the host's modem fail to receive a valid password, connection to the host system will be prohibited. The rationale un-
125 derlying the invention is thus that the terminal user need have no knowledge of the password(s), nor even of the host computer's telephone number if, for example, the terminal/host is a dedicated system, and thus a principal source for fraudulent ac-
130 cess is eliminated.

The user's end modem may also be used in a
conventinal data communications link, i.e. to a
non-protected system.
The system according to the invention can also
5 incorporate a callback facility as aforesaid so as to
further enhance the level of security provided by
the system. With hitherto disclosed port protection
devices incorporating a callback facility, entry of
the passwords is (to our knowledge) by manual
10 means; the present invention provides the facility
for automatic transmission of the password by the
user end modem. Further features which can be
provided in a system in accordance with the pres-
ent invention comprise the association of a status
15 code and/or a time-of-access zone with each valid
password. The status code can provide for imme-
diate access of a special status authorised caller to
the host computer thus bypassing the need for
callback to be effected, and the time-of-access zone
20 may be used to prevent an authorised user's ac-
cess to the host computer at times other than
those defined by his allocated time-of-access zone.
In accordance with yet a further aspect of the
present invention, in order to enable a callback
25 system to be utilised from any workstation location
and to be utilied by users, such as travelling sales-
persons for example, having mobile workstations
with no fixed location and a variable telephone
number, it is proposed that the host modem or
30 port protection device, in response to verification
of a received password transmitted by a user to-
gether with the user's current telephone number
location, generates a one-time short-term pass-
word and transmits it back to the user's location.
35 The user then has to re-dial the host computer and
can obtain access only by use of the one-time
short-term password within a predetermined short
time period of the original password entry. The re-
dialling of the host computer could be effected by
40 means of autodial equipment provided in the mo-
dem at the user's terminal end, the user's end mo-
dem receiving and temporarily holding the one-
time password transmitted by the host's modem;
by this means the need for user knowledge or con-
45 trol of passwords is completely removed thereby
enhancing security.
In the systems according to the invention, the
passwords, the user status codes and time-of-ac-
cess zones, and the callback telephone numbers, or
50 any of them, are not made accessible for modifica-
tion by the standard user; that is to say, such data
can be modified only at the command of an appro-
priately authorised key person at the host com-
puter location with such key person's access to the
55 host computer itself being password controlled.
Having thus described the concepts upon which
the present invention is based and recognising the
capability of the skilled technician in the data com-
munications art readily to put the herein-disclosed
60 inventive concepts into practical realisation without
need for further explanation, it is considered that
no further description of the present invention is
required herein. Various features, alterations and
modifications will occur to those possessed of ap-
65 propriate skills without departure from the spirit

and scope of the invention. Basically the invention
provides for security procedures to be completely
hidden from the user and involves no user inter-
vention.
70 As yet a further feature, the invention could
make use of encryption techniques for yet higher
levels of security.

CLAIMS

75 1. A password-protected data communication
system for transfer of data between remote user
terminals and a host computer via public tele-
phone networks or the like and wherein password
transaction(s) and/or interchange(s) are automati-
80 cally effected between special modems provided at
the user terminal and at the host computer without
action or intervention (other than call initiating ac-
tion) by the user who is denied access to or control
85 over the password(s).
2. A system in accordance with claim 1 includ-
ing a callback facility whereby, in response to re-
ception at the host modem of an acceptable
password, the host modem automatically seeks to
90 connect the host computer with a predetermined
user workstation location associated with the re-
ceived password.
3. A system in accordance with claim 2 wherein
the host modem, in response to verification of a
95 received password transmitted by a user together
with the user's current telephone number, gener-
ates a one-off short-term password and transmits it
back to the user's location, the user being enabled
to access the host computer only by utilisation of
100 such one-off short-term password within a prede-
termined limited time period.
4. A system in accordance with claim 3 wherein
the modem at the user's location is adapted and
arranged to automatically access the host com-
105 puter by utilisation of said one-off short-term pass-
word without intervention from the user.
5. A system in accordance with any of the pre-
ceding claims wherein the passwords utilised by
the system incorporate user status and/or time-of-
110 access zone codes.

Proceedings   Volume 2   **C1**

IEEE INFOCOM'95

The Conference on Computer Communications

Fourteenth Annual Joint Conference of the IEEE
Computer and Communications Societies

Bringing Information to People

April 2-6, 1995
Boston, Massachusettes

Sponsored by
IEEE Computer Society
IEEE Communications Society

The Use of Communications Networks to Increase Personal Privacy

N. F. Maxemchuk   *S. Low*

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

Abstract

Communications Networks can separate as well as join information. This ability can be used to increase personal privacy in an environment where advances in technology makes it possible to collect and correlate increasing amounts of information about individuals. The tools and principles necessary to increase personal privacy are demonstrated by creating an anonymous credit card, in which a person's identity and purchases are separated, and a national health insurance plan, in which treatment, payment and an individual's identity are separated. An analysis technique is developed to determine how well the information is separated.

1. Introduction

As computer memories have increased in size and decreased in cost, it has become reasonable to assemble vast amounts of information about individuals. As computer processors have become more powerful, it has become possible to correlate the information that is being assembled and to make inferences about individuals. As data networks have become ubiquitous and transmission rates have increased, it has become possible to combine the vast amounts of information that have been assembled in different locations, for different purposes.

Some of the uses of information can be annoying: Stores and credit card companies that sell information about an individual's purchases for directed advertising can significantly increase the mailings that we receive. Some of the uses of information have been made illegal: In January 1994 it was made illegal for video rental stores in New York State to sell lists of the movies that individuals rent. Some of the uses of information may change the way our society operates: In the Republican party rebuttal to the President's state of the Union message, in January 1994, one objection to a national health plan is the potential invasion of an individual's privacy.

Communications networks give us the ability to bring information together. They also give us the ability to separate and hide information. Some of the tools that make it possible to enhance privacy by communications are described in section 2. In section 2.1 a cryptographic protocol is described for communicating between two parties, and transferring trust between those parties, without either knowing the identity of the other. In section 2.2 an information analysis procedure is described that determines what information can be extracted when two or more parties collude. This analysis determines the effect when parties misbehave and also shows the worse that can happen when there are implementation errors.

Our objective is to control the access to information even though the information is needed and available in the network. The implementation of an anonymous credit card[1,2] is described in section 3. The information is separated and hidden so that a credit card company does not know its client's purchases or the stores that its client shops at, and a store does not know its customer's identity or the credit card company that has paid the bills. However, the company that extends credit knows its clients identity, the store knows what it has sold to a person, and there is a transfer of funds between the store and credit card company.

In the credit card system, anonymity is obtained while all of the capabilities of conventional credit cards, such as providing detailed billing and challenging purchases, are retained. It is also possible to issue the equivalent of an electronic subpoena to associate an individual's identity and purchases when there is reason to suspect illegal use of the payment mechanism. Currently, information about a person's purchases is available unless laws are passed to make that information private. With this mechanism, information about a person's purchases are private unless the law is used to make it available.

Three extensions of the basic anonymous credit card are digital cash, paying for network services, and increased privacy for electronic document distribution. When the mechanism is used for digital cash, section 3.3.1, there is no way to compromise a person's anonymity. However, this mechanism retains the protection against loss or theft of a credit card, and is more difficult to forge than conventional digital cash mechanisms. When paying for network services, section 3.3.2, this mechanism makes it possible for small

4d.1.1

venders, who aren't trusted to receive credit card numbers, to sell services. In electronic document distribution, section 3.3.3, the anonymous credit card is used to balance the interests of readers and publishers. Publishers cannot accumulate profiles on what a person reads, but can obtain the identities of people who illegally redistribute electronic documents.

An interesting extension of the anonymous credit card is to the the National Health Insurance Plan, in section 4. The straightforward application of the credit mechanism makes it possible to obtain anonymity when paying for services. In addition, it

— makes health records available anywhere, anytime, without disclosing a person's identity,

— makes it possible to conduct medical research on correlations between diseases and treatments without compromising the individuals involved, and

— allows an insurance company to monitor an individual's treatment without knowing who is being treated, unless they have illegally used the system.

As a result of considering the examples in this paper, the collusion analysis that was performed for the anonymous credit card has been related to the problem of finding paths in a graph, and generalized to take into account the difficulty or uncertainty in collusion. The generalized collusion analysis is described in section 5.

2. Tools

2.1 Double-Locked Box Protocol

The double-locked box provides communications between two users connected to different computers, or transfers funds between two accounts in two banks, without either computer, or bank, knowing the identity of the other. Only the bank that the account is located in knows the identity of the account, and only the computer that the user is connected to knows the identity of the user.

Communications passes through an intermediary, as was proposed by Chaum for untraceable electronic mail[3]. The message sender presents the computer he is connected to with a box that can only be opened by the intermediary. Inside the box is the identity of the destination computer and a second box that only that computer can open. Inside the second box is the identity of the message recipient.

When funds are transferred from an account in one bank to an account in a second bank, the intermediary operates as the federal reserve and trust is transferred as well as information. A funds transfer between account i in bank 1 to account j in bank 2 is demonstrated in figure 1. When bank 1 transfers amount M to an account specified by a double-locked box that the customer has

provided, it signs the message so that the federal reserve can verify that it is from a trusted bank. The federal reserve sends a signed message to bank 2 to deposit amount M in the account in the locked box. The federal reserve is responsible for settling the accounts between banks.

In a funds transfer environment it is particularly important to prevent messages from being replayed. For instance, if the destination account intercepted the message ordering the second bank to deposit funds, it might be tempted to have the deposit repeated. Two standard techniques to prevent replay attacks are sequence numbers and challenge response protocols. Either public keys or secret keys can be used to construct the funds transfer protocol, as described in references 1 and 2 respectively.

2.2 Collusion Analysis

Pieces of information about an individual are placed at different nodes in a network in order to make it difficult to associate the information. For instance, in the anonymous credit card we separate an individual's identity and purchases. Collusion analysis is used to determine how well the information is separated.

The players, such as the banks and intermediaries, associate information and messages. For instance, in figure 1, when funds are transferred from account i to account j

— bank 1 associates message A with all of the information in account i,

— the federal reserve associates bank 1, bank 2, message A, and message B, and

— bank 2 associates message B with all of the information in account j.

Messages A and B are unique. If bank 1 and the federal reserve collude they can combine all of the information each associated with message A. The source account becomes associated with bank 2 and message B. If the combination of bank 1 and the federal reserve then collude with bank B, they can combine all of the information each associates with message B. The source account and destination account become associated.

Collusion is dependent upon unique information. If two players have the same unique piece of information, and collude, then they can combine information. For instance, if there are a large number of accounts in each bank, banks 1 and 2 collude without the intermediary, and there is no unique information that is common accounts i and j, then there is no reason to associate the information in two of the accounts. However, if the two accounts have the same social security number associated with them, then the accounts can be associated.

4d.1.2
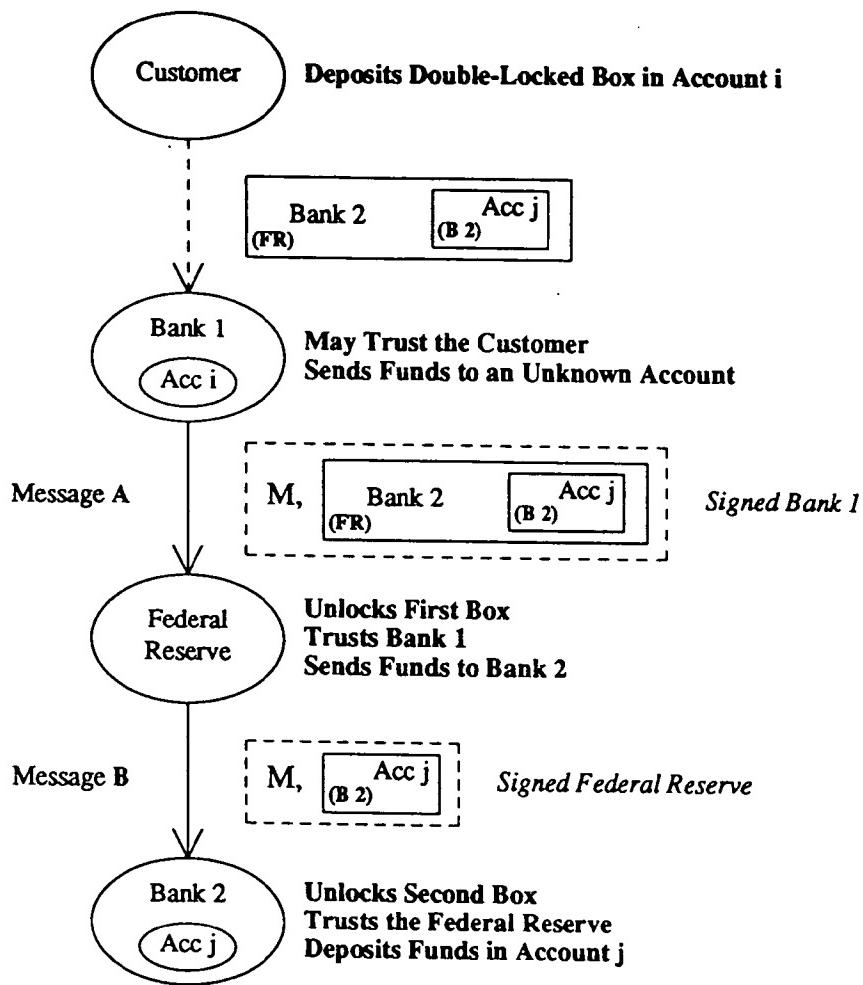
Figure 1. The Double-Locked Box Protocol

In a real system there is a large amount of information and a large number of messages. The amount of information that must be considered for collusion analysis can be reduced by combining similar information. When two pieces of information always appear jointly, there is no reason to consider them separately when performing collusion analysis. When several messages follow the same path, for the same purpose, then the ability to collude, in many instances, isn't improved by considering more than one. Even after combining information into common sets, the amount of information to be considered for collusion analysis may be large. In reference 2, collusion analysis is reduced to row reductions and multiplications on binary matrices. Automating this process makes it possible to consider more complex systems.

When performing collusion analysis, careful consideration must be given to what constitutes identifiable information. Suppose a bank logs a charge of $4.11 at 10:15 AM on 6/14/94, and a store logs a sale of the same amount, within a minute of the same time. Can this information be used to combine the information at the two locations? Depending upon what information is considered unique, different results are obtained from a collusion analysis.

3. Anonymous Credit Card

The anonymous credit card is implemented by constructing the electronic equivalent of a credit card company, a Swiss bank with anonymous accounts, a communication exchange, a federal reserve, and banks for stores, as shown in figure 2. The credit card company trusts an individual to repay a debt, and knows his identity. The store knows the merchandise that is purchased. The objective is to distribute the information so that a number of players must collude in order to

4d.1.3

associate an individual's purchases and identity. The detailed protocols, including the message formats that are transmitted, are described in reference 1. A higher level description of the protocol is presented here.

Credit is extended to the individual by the credit card company. The credit card company places credits in the individual's anonymous account using a double-locked box that the individual has placed in his credit card account. The bank with the anonymous account does not extend credit to the individual. This bank trusts the federal reserve, which trusts the credit card company. Therefore, the bank with the anonymous credit account does not need to know the individual's identity. At the end of the month, or when the credits are all used, the bank with the anonymous account presents a bill to the credit card company, using a double-locked box that the individual has deposited in his account. The credit card company presents the individual with a bill, and when it is paid, the credit card company deposits additional credits in the anonymous account.

An individual makes purchases in two phases, first he convinces the bank with the anonymous account that he is authorized to draw on that account, then he instructs that bank to transfer funds to the store's bank. Mechanisms that an individual can use to identify himself in the first phase, are described in references 1 and 2. In the second phase, funds are transferred from the bank with the anonymous account to the store's bank, using a double-locked box that the store gives to the customer with the bill. Once the funds transfer is complete, the store's bank notifies the store that it as been paid. Since the store does not trust the individual to pay a bill, the store does not need to know the individual's identity. The encryption and identification techniques required for the anonymous credit card have been implemented on a smart card.

Banks and credit card companies expect to make a profit when credit is extended. Either the store or customer must expect to pay for the use of the funds, and the service provided by the communication exchange. As funds flow through this system, each party can skim off a percentage or a fixed amount, depending upon what agreements have been reached.

3.1 Additional Services

The basic function performed by credit cards is to extend credit, pay the vendor for purchases, then to bill credit card holder. In the previous section we have described how this function is performed. However, we also expect to be able to:
— cancel lost cards,
— detect unusual spending patterns,
— receive itemized bills,
— return purchases, and
— challenge purchases that are charged to us.

Cancelling a lost cards is straightforward. We report the loss to our credit card company, which reduces the credit extended to our anonymous account to zero.

Two mechanisms are used to detect or prevent unusual use of the credit mechanism;
— the amount of credit transferred from the credit card company to the anonymous account is the maximum that can be spent, and
— rules can be placed in the anonymous account to limit the rate at which charges are made, or the maximum for a single charge.

Itemized bills are created, without disclosing the purchases, by allowing an individual to store a sales slip and personal note, encrypted with a key that only he can decrypt, along with the charges. The encrypted list is transferred form the anonymous account, to the credit card company, and then to the individual, along with the bill. When the digital sales slip is signed by the store, it provides a proof of purchase that the customer can use to return an item.

All of the messages that transfer funds are signed and are unique. These messages are saved for a period of time, as is currently required of funds transfer messages to banks. If a customer believes that a charge is erroneous, he can challenge it through the bank with the anonymous account. This bank has a signed message from the communications exchange authorizing the funds transfer. The bank presents this message to the communications exchange with an order from the customer to challenge the purchase. In order to prevent a bank from initiating a trace on its own, a separate organization issues these orders. The communications exchange keeps a log associating the message that it sent to the bank with the message that it received from the customer, that was signed by the customer's card. If an erroneous charge has been made, the bank with the anonymous account can trace the message that it sent to the store's bank to recover the funds.

A real concern with an anonymous payment mechanism is the ability to use it for illegitimate purposes. In this system, the linkages between customers and stores can be traced. If the equivalent of an electronic subpoena is issued against an individual, the billing messages can locate the anonymous accounts and the charging messages can locate the stores. Similarly, a subpoena against a store can force the store's bank to trace the messages transferring funds into the store's account back to the customers' anonymous accounts and from there back to the individuals. This is a secure tracking method, since the individuals cannot erase

4d.1.4
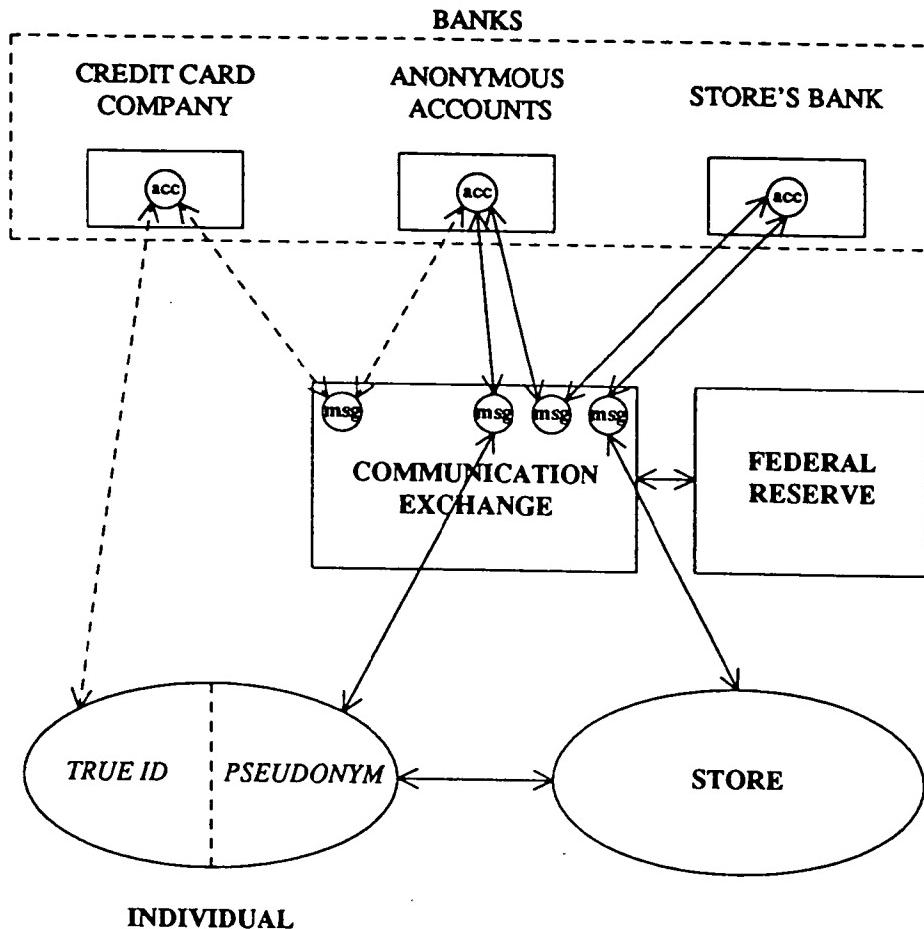
Figure 2. The Participants in an Anonymous Credit Card

messages in the trusted banks or the communications exchange.

3.2 Collusion

In the anonymous credit card, since the messages in the system are unique, it is always possible to collude along the message path linking two pieces of information. For instance, the store knows the purchases and the credit card company knows the customer's identity. If a store does not eavesdrop on the messages from its customers, the message path linking the store to the credit card company, as shown in figure 2, includes the store, the store's bank, the bank with the anonymous account, the credit card company, and the communications exchange. Depending upon whether or not the same communications exchange is used for all transactions, either five, six, or seven parties must collude to associate a customer's identity and purchases. If the store listens to the customer's messages, the

number of parties that must collude is reduced to four or five, the store, the bank with the anonymous account, the credit card company, and one or two communications exchanges.

The collusion path can be reduced if two parties share a unique piece of information. For instance, if the credit card system is small so that the time and amount of a purchase is unique, then the store and bank with the anonymous account, which both know this amount, can collude. The additional parties that must collude to learn the customer's identity are the communications exchange and the credit card company, so that the number of parties that must collude is reduced to four. When there are several banks with anonymous accounts, this is a weaker attack than following the message path because the store cannot be certain that it is colluding with the correct bank.

3.3 Related Applications

3.3.1 Digital Cash

The electronic funds transfer mechanisms developed for the anonymous credit card can be used as a replacement for digital cash. To construct a digital cash system from the anonymous credit card a user physically deposits funds in an anonymous account. Since credit is not extended to the individual, there is no need to know his identity. As a result, there is no way to associate a person's purchases and identity.

In most digital cash mechanisms[4,5,6,7,8,9] a user obtains sequences of bits that represent cash. Communications, processing and bookkeeping are required to make certain that a user does not duplicate the bits and spend them more than once. In addition, cryptographic techniques are relied upon to prevent a user from forging sequences of bits that are mistaken for cash.

When digital cash is implemented with the funds transfer mechanisms used in the anonymous credit card, the bits representing cash remain in trusted banks rather than being in a person's possession. The person cannot lose the bits. The bits cannot be stolen. And, the person cannot forge new bits.

The main disadvantage of the funds transfer mechanism is that it requires communications to make purchases. The more advanced digital cash mechanisms enable a user to "spend" the bits without communicating with a checking agency while the bits are being spent. At a later time the bits are sent to a central processing agency. If the bits are "spent" once, the spender is anonymous. If the bits are "spent" more than once, the spender's identity is revealed.

3.3.2 Paying for Network Services

In a commercial network environment, such as that being established on the Internet, electronic payment mechanisms are needed to make it possible for small businesses to start up.[10,11,12] It is not adequate to provide secure mechanisms to transfer credit card numbers. An individual may trust a large company, like Sear's, not to misuse a credit card number, but has no reason to trust an unknown individual. In addition, it is not always possible for new buisnesses to obtain mercantile accounts with credit card companies. In order to make it easier to start new businesses, electronic cash or credit mechanisms should be used to transfer funds.

Anonymity in a network environment is more difficult to obtain than when a person enters a store. When a person enters a store and carries his purchases out, there is no reason for the store owner to know the person's identity. However, if the individual wants the goods delivered, then he must disclose where he lives.

Unfortunately, a network is like the second case rather than the first. In order to become anonymous, the recipients network address must be obscured. Conventional ways to hide addresses on a network are to:

1. communicate through an intermediary that forwards messages from a source to a destination,
2. post information, encrypted with the recipients secret key, in a public place or broadcast it, or
3. place calls through a telephone company that is trusted not to provide "Caller ID."

It is possible to use the anonymous credit card without hiding network addresses. However, complete anonymity may be useful when there are a large number of small transactions between two parties. Instead of transferring funds for each transaction, a customer can transfer funds at the beginning of a session and trust the vendor to return the unused funds. If the user is anonymous, there is more reason for the vendor to be honest since the recipient may be a network checker.

3.3.3 Electronic Document Distribution

It has been noted that a major impediment to electronic document distribution is the relative ease with which electronic documents can be copied and redistributed. In order to protect a publisher's revenues, it has been proposed that each copy of an electronic document that a publisher distributes be made unique and registered to the individual who ordered it.[13,14] When illicit copies of a document are located, a publisher can use the unique characteristics to determine the original recipient.

A straightforward implementation of document marking requires publishers to create a reading profile for an individual. This invades an individual's privacy and, if the reading is work related, will concern employers. The anonymous credit card provides a means of balancing the interests of both the individuals and the publishers.

If articles are payed for with the anonymous credit card, the publisher doesn't need to know the purchaser's identity to discourage redistribution. The publisher can associate the message that verified that funds were transferred with the copy of the article. If multiple copies are located, a subpoena can be obtained to force the collusion needed to disclose the purchaser's identity.

4. National Health Insurance

The national health insurance plan presents an interesting application where an individual's right to privacy and society's need to control spending conflict. Most individuals feel that any medical or psychiatric treatment that they receive is between them and their physicians. Society has a responsibility to monitor any programs that it pays for, so that the programs are not

4d.1.6

abused. These apparently conflicting goals can be resolved by the privacy mechanisms that have been described.

The system can be set up as shown in figure 3. An individual has a personal account in a bank that knows his identity, an anonymous account with a health insurance company, and an anonymous account in a database of medical histories, as well as an anonymous credit card.

A person's complete medical history is stored in the anonymous account in the medical history database. Since the database is connected to the communications network, the medical history is available anywhere, anytime.

An employer, the government, or the individual deposits funds to pay insurance premiums into the personal account. Payments are made to the insurance company using the double-locked box protocol. The insurance company does not know the individual's identity, but has a double-locked box to the medical history account that is used to determine premiums and pay for medical services. It also has a double-locked box for the personal account, to present bills to the individual.

When a person acesses this database to make the information available to a health care provider, he uses a smart card and proves his identity the same way he does before making a purchase with the anonymous credit card. After treatment, the health care provider sends the information needed for insurance coverage and future treatment to the person's medical history account, using a double-locked box that it received with the patient information. Although funds are not transferred in this operation, trust is. The message from the health care provider is signed. The communications exchange verifies that the message is from a registered health care provider, and sends a message that it signs to the medical history database. The medical history database trusts the communications exchange to have checked the credentials of the health care provider, and enters the information into the account.

The entry that the health care provider makes in the medical history database also contains a double-locked box for his own bank account so that the insurance company will be able to pay his bill. After an individual receives medical treatment, he instructs his insurance company to pay for the service. The insurance company uses the double-locked box that accesses the individual's medical history file to obtain the bill and the double-locked box for the medical provider's account. The bill is certified by the communications exchange as being from an authorized health care provider. The insurance company pays for the covered services and the individual is made aware of any expenses that were not covered.

Presumably the remaining charges will be paid with the anonymous credit card, to maintain the patient's anonymity.

There is no direct link between a person's medical history and his identity. Insurance companies can audit medical history accounts on a regular basis without compromising an individual's right to privacy. If the insurance company suspects that an individual has misused his insurance policy, he must present the evidence to an agency that can authorize parties to collude in order to determine the individual's identity. The medical history database can also be used for medical research on correlations between diseases and treatments without compromising individual privacy.

The message path from an individual's medical history and his identity has four or five parties, the medical history database, the insurance company, the agency that maintains personal accounts, and one or two communications exchanges. The message path from an entry in a medical history to the health care provider has only the medical history database and the communication exchange. It is more difficult in this system to compromise an individual's privacy than that of a health care provider.

A model where an individual goes to a health care provider and proves his identity to obtain information stored in a medical history database is reasonable for many health care situations. It is not adequate for emergency situations when a person is unable to use, or does not have, his identification card. One means to obtain information in an emergency is to establish a server that associates biometric identifiers for individuals and double-locked boxes for medical history accounts. In an emergency the health care provider sends a signed message with the biometric identifier through a communications exchange to the emergency server. The communications exchange verifies that the health care provider is legitimate. The emergency server uses the double-locked box to request the medical history and forwards it to the health care provider.

5. Generalization of Collusion Analysis

As we have gone through the examples in this paper, it should be evident that collusion analysis is equivalent to finding the paths in a graph. The accounts in banks and the transactions in the intermediaries are nodes in the graph. Whenever there is information in two nodes that can be identified as belonging to the same individual, then a link exists between the nodes. The minimum number of parties that must collude to associate the two pieces of information is one plus the shortest path between the nodes with the information.

4d.1.7
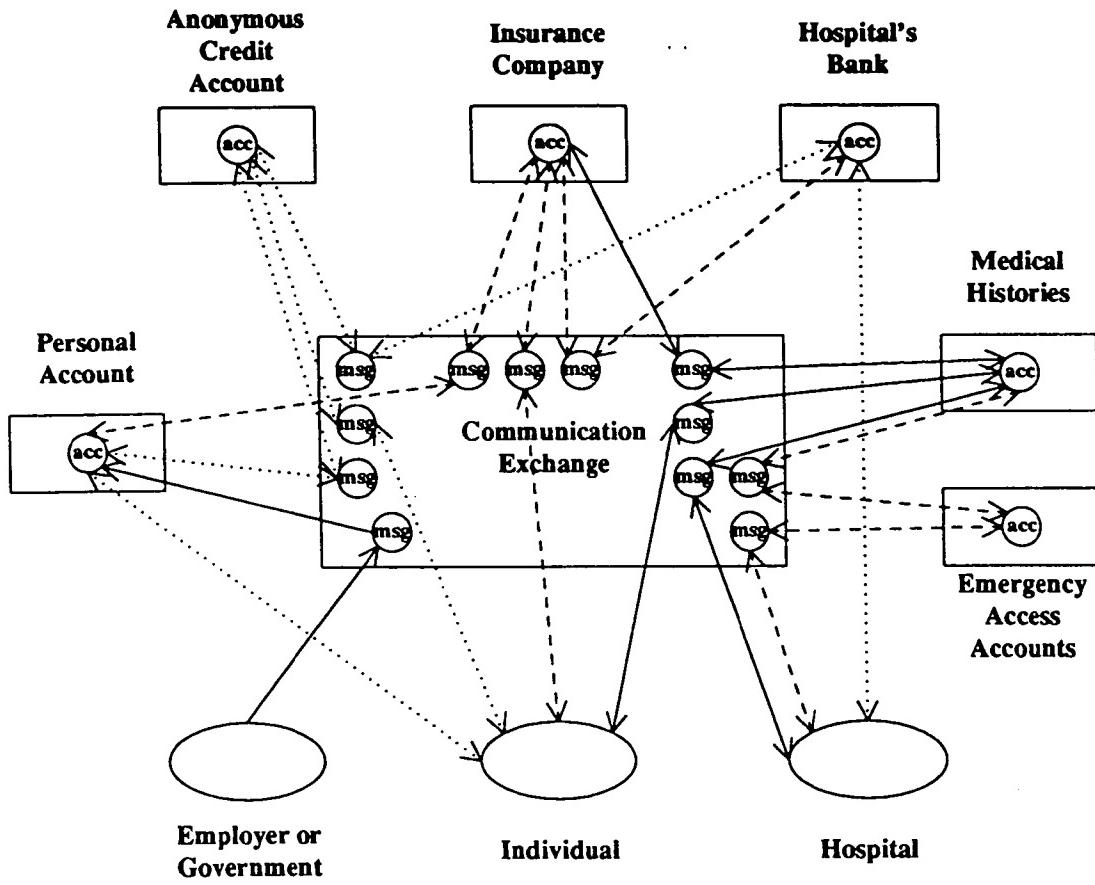
Figure 3. The Participants in the National Health Insurance

Certain messages in the applications must be unique. A graph with links corresponding to the unique messages determines the maximum number of parties that must collude. In order to achieve this upper bound, the rest of the information at the nodes is distributed so that no shorter paths are created. In the anonymous credit card and national health insurance applications, the upper bound can be realized.

The shortest path analysis is a first step toward performing collusion analysis, but does not provide the complete answer. The next step should take into account how difficult or how likely it is that unique, identifiable pieces of information can be used for collusion. In the message passing case, nodes are able to determine the other nodes that share the unique information. There are also instances in which multiple sites share a unique piece of information, but don't know the location of the other site, and instances in which the information isn't completely unique.

There are different degrees of difficulty in associating a message source and destination in communications networks. The funds transfer messages in the anonymous credit card are unique and identify the source and destination to the application. In the Internet,

packets are addressed so that the source must know the destination address. When an acknowledgement is expected, the destination must also know the source address. Information about the source and destination is available at the network layer, rather than in the application, and can be made less accessible to the user. When communications occurs in a switched network, such as the telephone network, the source and destination need not have access to each other's identity, but the network that established the circuit must. Collusion is more difficult because it involves another party.

When two sites have common information that can be used to collude, but the sites do not know each other's identity, collusion is more difficult. For instance, assume that a person has credit cards in two out of one hundred credit card companys, and each credit card company knows his social security number. If some of the credit card companies collude, the probability that the two companies with the common social security number are colluding is less than one. More parties must collude, on the average, to combine information than when the source and destination can identify each other.

When a piece of information at two sites is not unique, but has a probability of belonging to the same

individual, then a degree of collusion becomes possible. The ability to collude increases as the probability that the information belongs to the same individual increases. For instance, if an individual receives a credit card bill for a specific amount and an anonymous account issues a bill for the same amount, then there is a probability that they both apply to the same individual. If two successive bills that the anonymous account issues and the individual receives are the same, then the likelihood that the bills apply to the same individual increases. As the sequence of identical bills increases, so does the likelihood that they belong to the same individual. It is possible to collude at any point, but the confidence in the result of the collusion becomes higher as the probability increases.

Collusion analysis can be generalized to take into account the difficulty in obtaining information, the likelihood that useful sites will collaborate, or the uncertainty in information, by associating weights with the paths in the graphs. The generalized analysis indicates that collusion is easier when two parties that have unique information know each other's identity, than when they do not. Therefore, it is worth hiding the identities.

The techniques described in section 3.3.2, to hide the source and destination when transmitting information in a network, can be used to increase the difficulty of collusion. For instance, the broadcast mechanism can be used in a sysyem that requires acknowledgements if

— The source transmits to an unknown destination by encrypting a message with a key supplied by the user. The message is broadcast to all destinations, but only the destination with the decryption key can receive it.
— If the source supplies its own key in the message, then the destination can use the broadcast channel to acknowledge the message without knowing the source.

6. Conclusion

The tools presented in this work provide the means to design a range of applications that balance the rights of an individual to preserve his privacy, and the rights of society to protect the interests of the group. The proper balance between an individual's rights and society's needs are not addressed.

REFERENCES

[1] S. Low, N. F. Maxemchuk, S. Paul, "Anonymous Credit Cards," Proceedings of 2nd ACM Conference on Computer-Communications Security, Fairfax, Va. Nov. 2-4, 1994.

[2] S. Low, N. F. Maxemchuk, S. Paul, "The Anonymous Credit Card and Its Collusion Analysis," Submitted to IEEE Trans. on Networking.

[3] D. L. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms," Proceedings of Communications of the ACM, Vol. 24, No. 2, February 1981, pp. 84-88.

[4] D. Chaum "Security without Identification: Transaction Systems `to make Big Brother Obsolete" Communications of the ACM, October 1985, Vol. 28, No. 10, pp. 1030-1044.

[5] D. Chaum, "Privacy Protected Payments: Unconditional Payer and/or Payee Untraceability," Proceedings of Smart Card 2000, D. Chaum & I. Schaumuller-Bichl eds., North Holland, 1989, pp. 69-93.

[6] R. J. Anderson, "UEPS - A Second Generation Electronic Wallet," Proceedings of Computer Security - ESORICS 92, November 1992, pp.411-418.

[7] T. Okamoto, K. Ohta, "Universal Electronic Cash," Crypto '91, Santa Barbara, CA 11.-15., August 1991, Abstracts, 8.7-8.13, pp. 324-337.

[8] S. D'Amiano, G. Di Crescenzo, "Methodology for Digital Money Based on General Cryptographic Tools," Proceedings of Eurocrypt'94, pp. 151-162.

[9] T. Eng, T. Okamoto, "Single-Term Divisible Electronic Coins," Proceedings of Eurocrypt'94, pp. 311-323.

[10] S. Dukach, "SNPP: A Simple Network Payment Protocol," Proceedings of Computer Security Applications Conference, November 1992, pp. 173-179.

[11] G. Medvinsky, B. C. Neuman, "NetCash: A Design for Practical Electronic Currency on the Internet," Proceedings of the First ACM Conference on Computer and Communications Security, November 1993.

[12] M. A. Sirbu, "Internet Billing Server Prototype Scope Document," Carnegie Mellon University, Information Network Institute, INI Technical Report 1993-1, October 14, 1993.

[13] J. T. Brassil, S. Low, N. F. Maxemchuk, L. O'Gorman, "Electronic Marking and Identification Techniques to. Discourage Document Copying," IEEE Infocom '94, June 14-16, 1994, Toronto, Canada, pg. 1278-87.

[14] N. F. Maxemchuk, "Electronic Document Distribution," ATT Technical Journal, Sept. 1994, pg 73-80.

An attack on a recursive authentication protocol
A cautionary tale

P.Y.A. Ryan [a,*], S.A. Schneider [b]

[a] DRA, St. Andrews Rd., Malvern, Worcs WR14 3PS, United Kingdom
[b] Department of Computer Science, Royal Holloway, University of London, Egham, Surrey TW20 0EX, United Kingdom

Abstract

We describe an attack on a recursive authentication protocol proposed by John Bull of APM. The protocol is an implementation of a more abstract design that was analysed by Paulson and shown to establish session keys in a secure manner. The fact that Bull's implementation nevertheless fails to be secure in this sense provides an object lesson on how careful one has to be in interpreting the results of a formal analysis. © 1998 Elsevier Science B.V.

Keywords: Authentication; Security; Protocol vulnerabilities; Formal analysis

1. The protocol

In [1] Bull describes a recursive authentication protocol. Usually protocols involve a fixed number of participants and correspondingly fixed number of messages. This protocol can involve an indeterminate number of nodes and corresponding number of messages and as such poses interesting challenges to analysis. The protocol of [1] can have various uses but we will address one in particular suggested by Bull: to establish a chain of session keys between the nodes involved in the protocol run. Suppose the protocol is initiated by A and then goes through B, C and D before reaching the server S. Once the protocol has run to completion, new session keys K_{ab}, K_{bc}, K_{cd} and K_{ds} are established. K_{ab} should be known to exactly A and B (and possibly the server) and so on.

This should hold even if some nodes other than A, B or the server are malicious.

For illustration consider an instance of the protocol with just three nodes plus server. Initially each node has a private key it shares with the server. These are denoted K_a, K_b, etc. and serve to provide authentication and confidentiality.

We use the following notations: $hash_K\{X\}$ denotes a keyed hash with key K applied to X, $\{X\}_K$ denotes block encryption of X under K and $\oplus$ denotes the bitwise exclusive-or of two bit strings. Finally $Sign_k\{X\}$ is defined by

$$Sign_k\{X\} := X, hash_K\{X\}, \qquad (1)$$

i.e., the signing of data X using the hash keyed with key K. The signature can be verified by anyone also in possession of K.

A typical sequence of messages, in this case for 3 nodes, is as follows:

* Corresponding author.

1: $A \rightarrow B$: $Sign_{K_a}\{A, B, Na, -\}$
   let $Xa := Sign_{K_a}\{A, B, Na, -\}$
2: $B \rightarrow C$: $Sign_{K_b}\{B, C, Nb, Xa\}$
   let $Xb := Sign_{K_b}\{B, C, Nb, Xa\}$
3: $C \rightarrow S$: $Sign_{K_c}\{C, S, Nc, Xb\}$
4: $S \rightarrow C$: $A, B, Kab \oplus hash_{K_a}\{Na\}, \{A, B, Na\}_{Kab}$,
   $B, A, Kab \oplus hash_{K_b}\{Nb\}, \{B, A, Nb\}_{Kab}$,
   $B, C, Kbc \oplus hash_{K_b}\{Nb\}, \{B, C, Nb\}_{Kbc}$,
   $C, B, Kbc \oplus hash_{K_c}\{Nc\}, \{C, B, Nc\}_{Kbc}$
5: $C \rightarrow B$: $A, B, Kab \oplus hash_{K_a}\{Na\}, \{A, B, Na\}_{Kab}$,
   $B, A, Kab \oplus hash_{K_b}\{Nb\}, \{B, A, Nb\}_{Kab}$,
   $B, C, Kbc \oplus hash_{K_b}\{Nb\}, \{B, C, Nb\}_{Kbc}$
6: $B \rightarrow A$: $A, B, Kab \oplus hash_{K_a}\{Na\}, \{A, B, Na\}_{Kab}$

Note also that we have introduced local variables Xa and Xb to represent the outputs of nodes A and B respectively to give a more compact representation.

Each node takes its input, appends a nonce and some address information and signs the result using a hash keyed with its private key. It then outputs the result to the next node. The server thus eventually receives a message that consists of nested signatures. It can unravel these and so verify the path that the protocol has taken to reach it. The server then creates new session keys which it conceals by exclusive-oring them with hashes that only the intended recipients can compute. Consider for example the last line of the message send out by S, message 4 of the protocol. It is for C's benefit: it can compute the value of $hash_{K_c}\{Nc\}$ and so extract Kbc which it can verify by computing $\{C, B, Nc\}_{Kbc}$ and checking that this agrees with the value it has received. Similarly the middle line enables B to compute and verify Kbc and Kab.

The protocol makes extensive use of hash functions. Bull argues that this has advantages in terms of efficiency and sidesteps possible restrictions on the use of "strong" encryption. It does however appear to introduce a certain fragility that we will discuss further in the conclusions. The idea of using hash functions rather than encryption to perform authentication goes back to Gong [2].

2. The attack

This protocol is open to a rather easy attack, easy both in the sense that it is easy to spot and is easy to mount. We call it the domino attack for reasons that will be apparent below.

Assuming that the protocol is used to distribute a connected chain of keys linking all the nodes from the originator to the sever then the compromise of any one key leads to the compromise of all the keys. To see this observe that the server returns terms of the form

$$Kab \oplus hash_{K_a}\{Na\}, \quad Kab \oplus hash_{K_b}\{Nb\},$$

$$Kbc \oplus hash_{K_b}\{Nb\}, \quad \dots \quad Kls \oplus hash_{K_l}\{Nl\}. \quad (2)$$

Thus anyone can compute

$$Kab \oplus hash_{K_b}\{Nb\} \oplus Kbc \oplus hash_{K_b}\{Nb\}$$
$$= Kab \oplus Kbc \quad (3)$$

and similarly

$$Kbc \oplus Kcd \quad (4)$$

and so on.

Thus if any one of these keys are known, anyone monitoring the traffic can simply chain through these terms to derive all the others. Indeed, from the information it receives in the course of the protocol run, each node can calculate all the keys "downstream" from it. Hence the name domino attack. In particular the node next to the server actually gets everything it needs to compute all the keys.

The protocol should, on a successful run, provide the nodes with assurance that their keys are "good", i.e., known only to the intended principals, even if other nodes involved in the protocol are not trustworthy. Thus A and B should be confident that Kab is known only to them (and possibly the server) even if other nodes compromise their own keys, etc. We see that this requirement fails for this protocol.

3. The paradox

In [3], Paulson analyses a closely related protocol and shows that its key distribution is secure: that is the keys are private to their intended recipients even if other nodes are untrustworthy or compromised. Bull's protocol is an implementation of Paulson's. We thus have the apparent paradox of an implementation of a secure protocol being shown to be insecure.

The resolution of the paradox lies in the terms returned by the server. In Paulson's version the server returns terms of the form

$$\{A, Nb, Kab\}_{Kb}, \quad \{C, Nb, Kbc\}_{Kb}, \quad \text{etc.} \quad (5)$$

distribute ...
: nodes from the
mise of any one
he keys. To see
rms of the form

`Nb},

$ih_{kl}\{Nl\}$.   (2)

$\{Nb\}$

(3)

(4)

known, anyone
in through these
. from the infor-
ne protocol run,
s "downstream"
ck. In particular
ets everything it

ful run, provide
are "good",
...pals, even if
re not trustwor-
lent that Kab is
: server) even if
eys, etc. We see
tocol.

related protocol
s secure: that is
ecipients even if
promised. Bull's
ilson's. We thus
lementation of a
secure.
in the terms re-
rsion the server

tc.   (5)

in place of the terms involving the exclusive-or of the session keys with hash terms used in the Bull version, e.g.

$$Kab \oplus hash_{Kb}\{Nb\}, \quad Kbc \oplus hash_{Kb}\{Nb\}, \quad \text{etc.} \quad (6)$$

Paulson's analysis assumes that encryption is a primitive and so does not include any algebraic identities that would allow further terms, such as relationships between keys, to be deduced. In Bull's version these encryptions are implemented in a particular way that does give rise to additional identities and it is precisely these that are exploited in the attack.

4. The moral

This illustrates an important point about formal analysis of any system, and security protocols in particular: a formal model is necessarily an abstraction of the real system. As such it is always possible that some significant aspect of the real system is missed. Here the omission is rather subtle: Bull's choice of encryption introduces some algebraic identities between terms.

This problem is in fact an instance of a more general problem with developing secure systems that has long been recognised: that refinement, at least in its conventional forms, is about making systems more predictable. For most safety applications this is clearly appropriate. When dealing with security however, the more predictable the system the more information generally can be gleaned about its internals from limited observations. A detailed discussion can be found in [4].

Refinement can also be thought of in terms of theories: a specification corresponds to a theory and refinement corresponds to theory extension, in particular the adding of axioms. Theory extension preserves theorems: i.e., any theorem true in the original theory holds in the extended theory. Thus naively one would suppose that any result about the security of a system would be preserved under theory extension. The problem is that security is really a meta-theorem about the system, and meta-theorems are not preserved by theory extension. In this case the security result that we are trying to establish is that the spy cannot deduce certain facts. Adding to the axioms available to the

spy will increase the facts he can deduce and so can undermine the security property.

It is thus all too easy to be lured into a false sense of security by a formal analysis. This is not to say that formal analysis is useless but that you have to be very careful in interpreting its significance. This is a simple consequence of the definition of the word "formal" but is all too often overlooked.

An extensive and instructive discussion of the pitfalls arising from subtle interactions between properties of encryption algorithms and protocol designs can be found in [5].

It is interesting to consider how best to address this problem: given a successfully analysed model of a protocol and an implementation of this, what can we deduce about the security of the implementation? One obvious approach is to enhance the model to take account of the additional features of the implementation and repeat the analysis. This does not however seem to be a particularly efficient or pleasing solution. It would be more satisfying to be able to deduce the security of the implementation from the analysis of the model along with the relationship of the implementation to the model.

5. The fix

It is straightforward to fix the APM protocol to foil this style of attack: it suffices to ensure that the pairs of hashes have distinct values. For example the server could return terms of the form

$$Kab \oplus hash_{Kb}\{Nb, A\}, \quad Kbc \oplus hash_{Kb}\{Nb, C\}. \quad (7)$$

Indeed this is close in spirit to Paulson's version whilst retaining the use of hash functions to conceal the keys and requires minimal change to Bull's version. This certainly sidesteps the attack described above. It does however still allow some additional deductions not present in Paulson's analysis and so we cannot simply assume that this analysis applies. For example, the exclusive-or of pairs of hashes can be extracted:

$$hash_{Kb}\{Nb, A\} \oplus hash_{Ka}\{Na, B\} \quad (8)$$

though this seems not to lead to any vulnerability. Ideally this should be checked by a rigorous analysis.

6. Discussion

The protocol does suffer from other vulnerabilities. For example it suffers a failure of forward security: if the spy manages to break an old session key he can use this in conjunction with a replay attack to derive the value of a new session key and then use this to masquerade as another node.

Using our 3 node example, suppose that the spy has succeeded in breaking out the value of Kab, possibly long after it has been discarded. He can now replay message 1 from A to B and the protocol will run again but with the server generating a fresh set of keys (possibly going via a different set of nodes between B and the server but this is irrelevant). However as the spy has obtained Kab he can also compute the value of $hash_{Ka}(Na)$ and so when he sees the term $Kab^* \oplus hash_{Ka}(Na)$ from the server in the new run he can simply compute Kab^*, the new session key. As a result he can now masquerade to B as A. He will presumably intercept the message from B back to A to avoid A getting suspicious.

This attack is essentially the same as one described by Mathuria in [6] on an earlier but similar protocol and was drawn to our attention by Colin Boyd. A number of fixes suggest themselves: the server could inject nonces into its replies for example to prevent the same hash value from reappearing in the replay.

These vulnerabilities arise from the use of hash functions as a means of encryption and draw into question the merits of using hashes in this way. Paulson's version avoids both of these attacks.

It should also be noted that we have only addressed the key-distribution role of the protocol; if such a protocol were to be used to provide other security services these would have to be verified separately.

References

[1] J. Bull, The authentication protocol, APM Report, March 1997.

[2] L. Gong, Using one-way functions for authentication, ACM Comput. Comm. Review 19 (5) (1989) 8-11.

[3] L. Paulson, Mechanized proofs for a recursive authentication protocol, in: Proc. Computer Security Foundations Workshop (1997) 84-95.

[4] A.W. Roscoe, CSP and determinism in security modelling, in: Proc. IEEE Symp. on Security and Privacy (1995) 114-127.

[5] J.H. Moore, Protocol failures in cryptosystems, in: G.J. Simmons (Ed.) Contemporary Cryptography, The Science of Information Integrity, IEEE Press, Silver Spring, MD, 1992.

[6] A. Mathuria, Addressing weaknesses in two cryptographic protocols of Bull, Gong and Sollins, Electronics Lett. 31 (1995) 1543-1544.

Efficient and Timely Mutual Authentication

Dave Otway[†] and Owen Rees[‡]

The ANSA Project
24 Hills Road,
Cambridge CB2 1JP
United Kingdom
(ansa%alvey.uk@cs.ucl.ac.uk)

Abstract

This paper describes a protocol for efficient mutual authentication (via a mutually trusted third party) that assures both principal parties of the timeliness of the interaction without the use of clocks or double encipherment. The protocol requires a total of only four messages to be exchanged between the three parties concerned.

Introduction

This protocol is a development of the trusted third party, enciphered authentication protocols designed by Needham and Schroeder [1] and enhanced by Birrell [2]. It also eliminates the weakness pointed out by Denning and Sacco [3], whereby an intruder who discovers a conversation key can re-use the corresponding authenticator to initiate fraudulent conversations. Its development was prompted by an observation by Roger Needham that one of the basic principles of timely authentication was that "the suspicious party should always generate a challenge". In mutual authentication, both parties are suspicious of each other and of the freshness of the authentication messages; therefore each must generate independent challenges in order to assure themselves of the timeliness of the interaction. Such a challenge must be returned, as part of the authenticator, enciphered in the private key of the challenging party. Its important property is that it has not previously been used to authenticate the two parties concerned. This property can be guaranteed either by storing all previously used challenges, by using numbers from a monotonically increasing sequence (e.g. time) or probabilistically by generating a sufficiently large number randomly.

Prior knowledge and beliefs

Before authentication can take place, each party must be in possession of the following information:

first party:	$P_1 K_1 P_2$
second party:	$P_2 K_2$
authentication service:	$P_1 K_1 P_2 K_2$

where P_n is the name of the principal to which the nth party is currently affiliated and K_n is the private key of P_n.

Each party believes that private keys are known only to their corresponding principals and the trusted authentication service.

---

† Seconded from Marconi Instruments Ltd.   ‡ Seconded from Racal Information Technology Developments Ltd.

Protocol

A successful mutual authentication consists of the following sequence of messages (illustrated in figure 1), where $\{X\}^K$ means plaintext X enciphered in key K:

1) The first party sends the message $CP_1P_2\{R_1CP_1P_2\}^{K_1}$ to the second party. C is both a conversation identifier and a common challenge generated by the first party and which must be enciphered by both parties. R_1 is the first party's specific challenge and $\{R_1CP_1P_2\}^{K_1}$ is an enciphered request for the mutual authentication of P_1 and P_2.

   C must be in clear to allow the second party to encipher it. It may also be used to associate all the messages in the same authentication sequence and may be an identifier used in the underlying protocol layer.

   P_1P_2 must be in clear to identify the principals to the other parties, in particular, the authentication service needs them to look up the corresponding private keys.

2) The second party generates its own specific challenge R_2 and matching request $\{R_2CP_1P_2\}^{K_2}$, including the common challenge C. It then sends the message $CP_1P_2\{R_1CP_1P_2\}^{K_1}\{R_2CP_1P_2\}^{K_2}$ to the authentication service.

3) The authentication service looks up K_1 and K_2 using P_1 and P_2, then deciphers both requests and verifies that they form a matching pair (i.e. both contain CP_1P_2). If so, it chooses a conversation key K_C and returns the reply $C\{R_1K_C\}^{K_1}\{R_2K_C\}^{K_2}$, containing a matching pair of enciphered authenticators, identified by C, to the second party.

4) The second party deciphers the second authenticator $\{R_2K_C\}^{K_2}$ using its private key K_2 to obtain its own challenge R_2 and the conversation key K_C, then forwards the reply $C\{R_1K_C\}^{K_1}$, containing the first authenticator, identified by C, to the first party.

   The first party deciphers the first authenticator $\{R_1K_C\}^{K_1}$ using its private key K_1 to obtain its own challenge R_1 and the conversation key K_C.
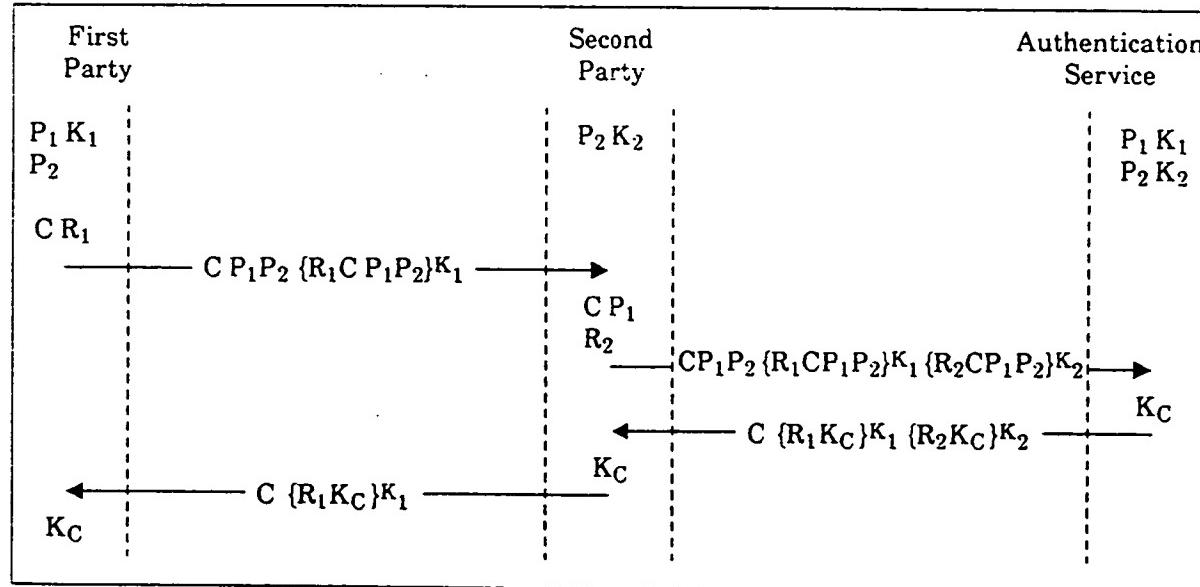

Figure 1

Subsequent knowledge and beliefs

The authentication service is able to decipher both authentication requests $\{R_1 CP_1 P_2\}^{K_1}$ and $\{R_2 CP_1 P_2\}^{K_2}$ using the private keys of the alleged principals. If they match (i.e. both contain $CP_1 P_2$), then it knows that each was generated by some party which knew the private key of the corresponding principal and that both parties wished to converse with each other (because of $P_1 P_2$) at some coincident time (because of C). The authentication service does not know whether a malicious replay of the whole message has taken place, it simply issues information that is of no value to anyone but the two genuine parties.

When the second party has decoded its authenticator $\{R_2 K_C\}^{K_2}$ using its private key K_2, it knows that if R_2 matches its original challenge then the reply is timely and was generated by the authentication service. Because it trusts the authentication service only to issue a matching pair of authenticators if the original pair of requests were genuine and matched, it accepts P_1 as the first party's principal and K_C as the secret conversation key.

When the first party has decoded its authenticator and verified R_1 then it also knows the reply is timely and was generated by the authentication service. Because it has the same beliefs about the authentication service as the second party it accepts P_2 as the second party's principal and K_C as the secret conversation key.

Because the first party chose the common challenge C, it knows that the second party's request must have been timely for the authentication service to verify that the two requests matched, but the second party still has no assurance that the original request from the first party was not a replay. However, both parties should now be in possession of the secret conversation key K_C, and the prompt receipt of a message correctly enciphered in KC assures the second party that the first party really is in possession of K_1 and must therefore be affiliated to P_1.

Discussion

The authentication requests and replies are symmetrical with respect to the two principals. This property enables either party to subsequently initiate a re-authentication and change of conversation key.

The authentication reply messages are not re-usable because they would no longer be timely; therefore the full protocol must be used when restarting a conversation after one of the parties has discarded the conversation key. The small number of messages required makes this acceptable, especially since both parties are assured of the timeliness of the new authenticators and the new key.

The authentication service is not obliged to keep state relating to active conversations and can therefore avoid the scaling problems that this would cause. However, it is able to detect and log fraudulently constructed requests.

The first and last messages of this protocol can be piggy-backed onto the opening exchange of a connection-oriented conversation, as can the exchange of a cipher initialization vector.

Known plaintext $CP_1 P_2$ being enciphered in the private keys K_1, K_2 is no longer a problem with modern stream ciphers, especially if it is always preceded by a random number.

Acknowledgment

References

1. NEEDHAM, R. M., and SCHROEDER, M. D. Using encryption for authentication in large networks of computers. *Commun. ACM 21*, 12 (Dec. 1978), 993-999.

2. BIRRELL, A. D. Secure communication using remote procedure calls. *ACM Trans. Computer Systems 3*, 1 (Feb 1985), 1-14.

3. DENNING, D. E., and SACCO, G. M. Timestamps in key distribution protocols. *Commun. ACM 24*, 8 (Aug. 1981), 533-536.

Mechanized Proofs for a
Recursive Authentication Protocol

Lawrence C. Paulson
Computer Laboratory
University of Cambridge
Pembroke Street
Cambridge CB2 3QG
England

lcp@cl.cam.ac.uk

Abstract

A novel protocol has been formally analyzed using the prover Isabelle/HOL, following the inductive approach described in earlier work [9]. There is no limit on the length of a run, the nesting of messages or the number of agents involved. A single run of the protocol delivers session keys for all the agents, allowing neighbours to perform mutual authentication. The basic security theorem states that session keys are correctly delivered to adjacent pairs of honest agents, regardless of whether other agents in the chain are compromised. The protocol's complexity caused some difficulties in the specification and proofs, but its symmetry reduced the number of theorems to prove.

1. Introduction

Security protocols are notoriously prone to error. One problem is the combinatorial complexity of the messages that an intruder could generate. Another, quite different, problem is that of specifying precisely what the protocol is to accomplish: proof of identity, session key distribution, establishment of shared secrets, etc.

Researchers are developing methods of analyzing protocols formally—either to search for attacks [6] or to prove correctness properties [2, 4, 7]. Recently, I have announced a new proof method, based on inductive models of protocols and an intruder [9]. Unlike model-checking approaches, it imposes no restrictions for the sake of finiteness. The automated provers in Isabelle/HOL [10] let the user analyze a typical protocol in a few working days. Below I shall describe an application of the method to an unusual, variable-length protocol.

The Otway-Rees protocol [8], which assumes a shared-key setting, allows an agent A to establish a session with some other agent, B. An authentication server generates a fresh session key Kab and distributes it to A and B. This protocol is widely accepted as correct; in previous work, I have proved basic guarantees for each party. At the end of a run, the session key just received is known only to the server and the other party, provided both A and B are uncompromised. Like many similar protocols, it has a fixed number of steps and all messages have simple, fixed formats.

The *recursive authentication protocol* [3] generalizes Otway-Rees to an arbitrary number of parties. First, A contacts B. If B then contacts the authentication server then the run resembles Otway-Rees. But B may choose to contact some other agent C, and so forth; a chain of arbitrary length may form. During each such round, an agent adds its name and a fresh nonce to an ever-growing request message.

For the sake of discussion, suppose that C does not extend the chain but instead contacts the authentication server. The server generates fresh session keys Kab and Kbc: in the general case, one key for each pair of agents adjacent in the chain. It prepares two certificates for each session key: one for each party. It gives the bundle of certificates to the last agent (C). Each agent takes two certificates and forwards the remainder to its predecessor in the chain. Finally, A receives one certificate, containing Kab.

Such a protocol is hard to specify, let alone analyze. Neither the number of steps, nor the number of parties, nor the number of session keys are fixed in advance. The server's response to the agents' accumulated requests cannot be given as a simple pattern; it requires a recursive program.

Even which properties to prove are not obvious. One might simplify the protocol to distribute a single session key, common to all the agents in the chain. But then, security

between A and B would depend upon the honesty of C, an agent possibly not known to A. There may be applications where such a weak guarantee might be acceptable, but it seems better to give a separate session key to each adjacent pair. I have proved a general guarantee for each participant: if it receives a certificate containing a session key and the name of another agent, then only that agent (and the server) can know the key. (In the sequel, "server" will always mean "authentication server.")

The paper describes the protocol in detail (§2.). It reviews the inductive approach to protocol analysis (§3.) and describes how it was extended with hashing (§4.). It presents the formal model of the protocol (§5.) and describes the main results proved (§6.). It discusses possible attacks on the protocol (§7.) and offers a few conclusions (§8.).

2. The Recursive Authentication Protocol

The protocol was invented by John Bull of APM Ltd., who (in a private communication) describes its objectives as follows:

> The project is exploring a model of security where application components are in control of security policy and its enforcement. The novelty of the approach is that the infrastructure no longer provides security for the applications, but provides them with the means to defend themselves.

The description below uses traditional notation, slightly modified. Let Hash X be the hash of X and Hash$_X Y$ the pair $\{$Hash$\{X, Y\}, Y\}$. Typically, X is an agent's long-term shared key and Hash$\{X, Y\}$ is a message digest, enabling the server to check that Y originated with that agent. Figure 1 shows a typical run, omitting the hashing.

Agent A starts a run by sending B a request:

$$1.\ A \to B : \text{Hash}_{Ka}\{A, B, Na, -\}$$

Here Ka is A's long-term shared key, Na is a fresh nonce, and $(-)$ is a placeholder indicating that this message started the run. In response, B sends something similar but with A's message instead of the placeholder:

$$2.\ B \to C : \text{Hash}_{Kb}\{B, C, Nb, \text{Hash}_{Ka}\{A, B, Na, -\}\}$$

Step 2 may be repeated as many times as desired. Each time, new components are added to the message and a new message digest is prefixed. The recursion terminates when some agent performs step 2 with the server as the destination.

In step 3, the server prepares session keys for each caller-callee pair. It traverses the accumulated requests to build up its response. If (as in §1.) the callers were A, B and C in that order, then the final request is

$$\text{Hash}_{Kc}\{C, S, Nc, \text{Hash}_{Kb}\{B, C, Nb, \text{Hash}_{Ka} \cdots\}\}.$$
$$\uparrow \qquad\qquad\qquad \uparrow$$

The arrows point to the occurrences of C, which appear in the outer two levels. C has called S (the server) and was called by B. The server generates session keys Kcs and Kbc and prepares the certificates $\{Kcs, S, Nc\}_{Kc}$ and $\{Kbc, B, Nc\}_{Kc}$. The session key Kcs is redundant because C already shares Kc with the server. Including it allows the last agent in the chain to be treated like all other agents except the first: the initiator receives only one session key.

Having dealt with C's request, the server discards it. Looking at the remaining outer two levels, the request message is

$$\text{Hash}_{Kb}\{B, C, Nb, \text{Hash}_{Ka}\{A, B, Na, -\}\}.$$
$$\uparrow \qquad\qquad\qquad \uparrow$$

The server now prepares two certificates for B, namely $\{Kbc, C, Nb\}_{Kb}$ and $\{Kab, A, Nb\}_{Kb}$. Note that Kbc appears in two certificates, one intended for C (containing nonce Nc and encrypted with key Kc) and one for B.

At the last iteration, the request message contains only one level:

$$\text{Hash}_{Ka}\{A, B, Na, -\}.$$
$$\uparrow$$

The $(-)$ token indicates the end of the requests. The server generates one session key and certificate for A, namely $\{Kab, B, Na\}_{Ka}$.

In step 3 of the protocol, the server replies to the request message by returning a bundle of certificates. In our example, it would return five certificates to C.

$$3.\ S \to C : \{Kcs, S, Nc\}_{Kc}, \{Kbc, B, Nc\}_{Kc},$$
$$\{Kbc, C, Nb\}_{Kb}, \{Kab, A, Nb\}_{Kb},$$
$$\{Kab, B, Na\}_{Ka}$$

In step 4, an agent accepts the first two certificates and forwards the rest to its predecessor in the chain. Every agent performs this step except the one who started the run.

$$4.\ C \to B : \{Kbc, C, Nb\}_{Kb}, \{Kab, A, Nb\}_{Kb},$$
$$\{Kab, B, Na\}_{Ka}$$
$$4'.\ B \to A : \{Kab, B, Na\}_{Ka}$$

The description above describes a special case: a protocol run with three clients. The conventional protocol notation cannot cope with arbitrary numbers of participants, let alone recursive processing of nested messages. Section §5. below will specify the protocol as an inductive definition.

My version of the protocol differs from the original in several respects:

- The dummy session key Kcs avoids having to treat the last agent as a special case. All agents except the first

The figure shows circles labeled S, A, B, C with arrows and labels:

Between S and C (upward):
C,S,Nc,
B,C,Nb,
A,B,Na,-

Between S and C (downward):
{Kcs,S,Nc}Kc,
{Kbc,B,Nc}Kc,
{Kbc,C,Nb}Kb,
{Kab,A,Nb}Kb,
{Kab,B,Na}Ka

Between A and B:
A,B,Na,-
{Kab,B,Na}Ka

Between B and C:
B,C,Nb,
A,B,Na,-
{Kbc,C,Nb}Kb,
{Kab,A,Nb}Kb,
{Kab,B,Na}Ka
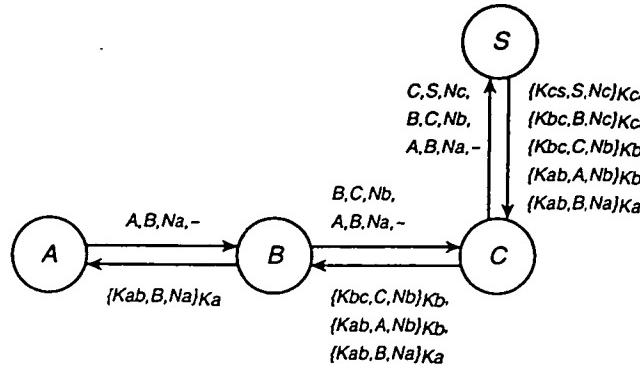
Figure 1. The Recursive Authentication Protocol with Three Clients

take two certificates. An implementation can safely omit the dummy certificate. Removing information from the system makes less information available to an intruder.

- In the original protocol, an agent's two certificates were distinguished only by their order of arrival; an intruder could easily exchange them. To correct this flaw, I added the other party's name to each certificate. Such explicitness is recommended as good engineering practice [1]. It also simplifies the proofs; a similar change to the Otway-Rees protocol cut the proofs in half [9]. Bull and Otway have accepted my change to their protocol [3].

- The original protocol implements encryption using exclusive "or" (XOR) and hashing. For verification purposes, encryption should be taken as primitive. Correctness of the protocol does not depend upon the precise form of encryption, provided it implemented properly. The original use of XOR turned out to be flawed (see §7.).

- Protocol messages contain some information that is important for engineering purposes but logically redundant. Omitting such information both simplifies and strengthens the proofs. Adding redundancy to a safe protocol cannot make it unsafe.

3. Review of the Inductive Approach

Informal safety arguments, which involve reasoning that dangerous states are unreachable, are made rigorous using induction. Protocols are modelled in standard predicate calculus and set theory.[1] A protocol specifies a set of possible

---

traces. Each trace is a list of events and may contain numerous runs, including interleaved and aborted runs. Each event has the form Says A B X, which represents the attempt by A to send B the message X.

One simplification in the model is the lack of an event to represent the reception of a message. We cannot assume that each message reaches its destination; we cannot identify the sending of a message in step i with the receipt of that message. But we can identify the sending of a message in step $i + 1$ with the receipt of a satisfactory message from step i. The model describes what may happen but never forces an agent to respond to any message. Thus, the model identifies the following circumstances:

- Message X was intercepted before it could reach B.

- Message X reached B, but B was down.

- Message X reached B, but B declined to respond.

An inductive definition consists of a set of rules. Modelling a protocol requires one rule for each protocol step. A typical protocol step $A \rightarrow B : X$ requires a rule saying that an existing trace can be extended with the event Says A B X. If the step is the response to another message Y, then the rule will be subject to the condition Says B' A Y. Other conditions may refer to messages that A has already sent earlier in the same run, typically for the purpose of confirming that a challenge has triggered an adequate response.

Agent names such as A and B are variables ranging over all agents, though rules frequently have conditions such as $A \neq B$ or $A \neq S$.

An additional rule models the attacker, Spy. He cannot crack ciphers by brute force, but has somehow got hold of some agents' long-term keys. He reads all traffic, decrypting messages using keys he holds and accumulating everything so obtained. At any point, he may send spoof messages composed using the data at his disposal. (Interception of messages does not have to be modelled explicitly, as remarked above.) He is accepted by the others as an honest agent.

To model the spy's capabilities, three operators are defined on sets of messages.

- parts H is the set of all components of H that are potentially recoverable, perhaps using additional keys.

- analz H is the set of all messages that can be decrypted from H using only keys obtainable from H.

- synth H is the set of all messages that can be built up from H.

These operators are themselves defined inductively and satisfy numerous useful laws. The spy draws spoof messages from the set synth(analz H), where H includes the history of past traffic and the spy's initial knowledge.

4. A Formalization of Hashing

Hitherto, I have considered messages to be built from agent names, nonces and keys by concatenation and encryption. The recursive authentication protocol assures integrity by means of hashing. Encryption could be used instead, but hashing is easily added to the model.

The datatype msg now admits messages of the form Hash X, where X is a message. Like all the message primitives, Hash is assumed to be injective (collision-free). The operators analz and synth treat hashing in the obvious way. The definition of analz requires no change; the effect is to say that nothing can be decrypted from Hash X. For synth, we insert the rule

$$X \in \text{synth}\, H \implies \text{Hash}\, X \in \text{synth}\, H,$$

allowing hashing to be used freely when composing spoof messages.

A question arises concerning the treatment of hashing by parts: is X a part of Hash X? The protocol involves messages of the form Hash$\{Ka, X'\}$. A *yes* answer would imply $Ka \in$ parts H, even if Ka were uncompromised; we could no longer reason about the security of long-term keys in the normal way [9]. A *no* answer seems right. It causes parts H to return the items *potentially recoverable* from H, which is a subset of the *ingredients* of H. Contrast the tasks of recovering the mainspring from a watch and recovering the eggs from a cake. An "ingredients" operator is not needed just now.

The new laws concerning Hash X resemble those for other atomic data, such as nonces. Here, ins X H is the set $\{X\} \cup H$.

$$\text{parts}(\text{ins}(\text{Hash}\, X)\, H) = \text{ins}(\text{Hash}\, X)(\text{parts}\, H)$$
$$\text{analz}(\text{ins}(\text{Hash}\, X)\, H) = \text{ins}(\text{Hash}\, X)(\text{analz}\, H)$$
$$\text{Hash}\, X \in \text{synth}\, H \implies \text{Hash}\, X \in H \lor X \in \text{synth}\, H$$

These laws state that Hash X contributes nothing other than itself to the result of parts or analz. The addition of Hash as a new message form does not invalidate any of the 90 or so laws concerning parts, analz and synth.

The Hash$_X$ Y notation is trivially defined in Isabelle:

$$\text{Hash}[X]Y \equiv \{\text{Hash}\{X, Y\}, Y\}.$$

Requests in the protocol have the form Hash$_X$ Y, where Y may contain another request. Rewriting a request by the definition of Hash$_X$ Y would cause exponential blowup. Instead, we can apply laws that treat Hash$_X$ Y as a primitive. The following law concerns parts; an analogous one holds for analz:

$$\text{parts}(\text{ins}(\text{Hash}_X\, Y)\, H)$$
$$= \text{ins}(\text{Hash}_X\, Y)(\text{ins}(\text{Hash}\{X, Y\})(\text{parts}(\text{ins}\, Y H)))$$

A further law is subject to $X \notin \text{synth}(\text{analz}\, H)$, as when X is an uncompromised long-term key:

$$\text{Hash}_X\, Y \in \text{synth}(\text{analz}\, H) \iff$$
$$\text{Hash}\{X, Y\} \in \text{analz}\, H \land Y \in \text{synth}(\text{analz}\, H)$$

This law says that the message Hash$_X$ Y can be spoofed iff Y can be and a suitable message digest is available (an unlikely circumstance). Blowup can still occur using such laws, but it is no longer inevitable. A formula such as Nonce $N \in$ parts H will simplify to the obvious outcome.

5. Modelling the Protocol

For the most part, this protocol is modelled just like the fixed-length protocols considered previously [9]. The inductive definition rules for the empty trace and the spy are standard. The other rules can be paraphrased as follows:

1. If *evs* is a trace, Na is a fresh nonce and B is an agent distinct from A and S, then we may add the event

   Says A B (Hash$_{\text{shrK}\, A}\{A, B, Na, -\}$).

   A's long-term key is written shrK A. For the token $(-)$ I used the name S, but any fixed message would do as well.

2. If *evs* is a trace containing the event Says A' B Pa, where $Pa = \{Xa, A, B, Na, P\}$, and Nb is a fresh nonce and $B \neq C$, then we may add the event

   Says B C (Hash$_{\text{shrK}\, B}\{B, C, Nb, Pa\}$).

   The variable Xa is how B sees A's hash value; he does not have the key needed to verify it. Component P might be $(-)$ (if A started the run) or might have the

same form as Pa, nested to any depth. Agent C might be the server or anybody else.

All the proofs about the protocol become simpler if the equation $Pa = \cdots$ is never applied. The proofs therefore hold of a weaker protocol in which any agent may react to any message by sending an instance of step 2. Ill-formed requests may result, but the server will ignore them.

3. If evs is a trace containing the event Says B' S Pb, and $B \neq S$, and if the server can build from request Pb a response Rb, then we may add the event

$$\text{Says } S \ B \ Rb.$$

The construction of Rb includes verifying the integrity of Pb; this process is itself defined inductively, as we shall see. The rule does not constrain the agent B, allowing the server to send the response to anybody. We could get the right value of B from Pb, but the proofs do not require such details.

4. If evs is a trace containing the two events

$$\text{Says } B \ C \ (\mathsf{Hash}_{\mathsf{shrK}\, B}\{B, C, Nb, Pa\})$$
$$\text{Says } C' \ B \ \{\mathsf{Crypt}(\mathsf{shrK}\ B)\{Kbc, C, Nb\},$$
$$\mathsf{Crypt}(\mathsf{shrK}\ B)\{Kab, A, Nb\}, \ R\}$$

and $A \neq B$, then we may add the event

$$\text{Says } B \ A \ R.$$

B decrypts the two certificates, compares their nonces with the value of Nb he used, and forwards the remaining certificates (R).

The final step of the protocol is the initiator's acceptance of the last certificate, $\mathsf{Crypt}(\mathsf{shrK}\ A)\{Kab, B, Na\}$. This step need not be modelled since A makes no response.

For many protocols, an "oops" message can model accidental loss of session keys. One then proves that an old, compromised session key cannot later become associated with new nonces [9]. An oops message cannot easily be expressed for the recursive authentication protocol because a key never appears together with both its nonces. Despite the lack of an oops message, the spy can get hold of session keys using the long-term keys of compromised agents; I trust the model is adequately realistic.

5.1. Modelling the Server

The server creates the list of certificates according to another inductive definition. It defines not a set of traces but a set of triples (P, R, K) where P is a request, R is a response and K is a session key. Such triples belong to the

set respond evs, where evs (the current trace) is supplied to prevent the reuse of old session keys. Component K returns the newest session key to the caller for inclusion in a second certificate.

The occurrences of Hash in the definition ensure that the server accepts requests only if he can verify the hashes using his knowledge of the long-term keys. The inductive definition consists of two cases.

1. If Kab is a fresh key (that is, not used in evs) then

$$(\ \mathsf{Hash}_{\mathsf{shrK}\, A}\{A, B, Na, -\},$$
$$\mathsf{Crypt}(\mathsf{shrK}\ A)\{Kab, B, Na\},$$
$$Kab) \ \in \ \text{respond } evs.$$

This base case handles the end of the request list, where A seeks a session key with B.

2. If $(Pa, Ra, Kab) \in$ respond evs and Kbc is fresh (not used in evs or Ra) and

$$Pa = \mathsf{Hash}_{\mathsf{shrK}\, A}\{A, B, Na, P\}$$

then

$$(\ \mathsf{Hash}_{\mathsf{shrK}\, B}\{B, C, Nb, Pa\},$$
$$\{\mathsf{Crypt}(\mathsf{shrK}\ B)\{Kbc, C, Nb\},$$
$$\mathsf{Crypt}(\mathsf{shrK}\ B)\{Kab, A, Nb\}, \ Ra\},$$
$$Kbc) \ \in \ \text{respond } evs.$$

The recursive case handles a request list where B seeks a session key with C and has himself been contacted by A. The respond relation is best understood as a pure Prolog program. Argument Pa of (Pa, Ra, Kab) is the input, while Ra and Kab are outputs. Key Kab has been included in the response Ra and must be included in one of B's certificates too.

An inductive definition can serve as a logic program. Because the concept is Turing powerful, it can express the most complex behaviours. Yet, such programs are easy to reason about.

5.2. A Coarser Model of the Server

For some purposes, the respond evs relation is needlessly complicated. Its input is a list of n requests, for $n > 0$, and its output is a list of $2n + 1$ certificates. Many routine lemmas hold for any list of certificates of the form $\mathsf{Crypt}(\mathsf{shrK}\ B)\{K, A, N\}$. The inductive relation responses evs generates the set of all such lists. It contains all possible server responses and many impossible ones.

The base case is simply $(-) \in$ responses evs and the recursive case is

$$\{\mathsf{Crypt}(\mathsf{shrK}\ B)\{K, A, N\}, \ R\} \in \text{responses } evs$$

if $R \in$ responses evs and K is not used in evs.

In secrecy theorems (those expressed in terms of analz), each occurrence of Crypt can cause a case split, resulting in a substantial blowup after simplification. Induction over responses introduces only one Crypt, but induction over respond introduces three. Of course, responses includes some invalid outputs; some of the main theorems can only be proved for respond.

5.3. Alternative Formats for Certificates

In the original protocol, B took two certificates of the form Crypt(shrK B){K, Nb}. He inferred the name of the other key-holder from the certificates' arrival order, which an attacker could change. This vulnerability had to be corrected. I experimented with putting both session keys into a single certificate of the form

$$\text{Crypt(shrK } B)\{Kab, Kbc, Nb\}.$$

An enemy can no longer exchange the keys. But each session key still appears in two certificates; Kbc would also appear in

$$\text{Crypt(shrK } C)\{Kbc, Kcd, Nc\}.$$

This format would have required two sets of proofs: one concerning the first session key in a certificate and one concerning the second. I therefore took Abadi and Needham's advice and chose certificates of the form

$$\text{Crypt(shrK } A)\{Kab, B, Na\}.$$

Now, permuting the certificates can do no harm: each quotes the name of the other holder of the session key. Moreover, this format preserves the protocol's symmetry.

6. Main Results Proved

For the most part, the analysis resembles that of the Otway-Rees protocol. Simple consistency checks, or "possibility properties," are proved first. Regularity lemmas come next: elementary facts such as that the long-term keys of uncompromised agents never form part of any message. (They do form part of hashed messages, however; recall the discussion in §4. above.) Secrecy theorems govern the use of session keys, leading to the main guarantee: if the certificate Crypt(shrK A){Kab, B, Na} appears as part of any traffic, where A and B are uncompromised, then Kab will never reach unintended parties. Another theorem guarantees that such certificates (for uncompromised agents) originate only with the server.

Possibility properties do not express liveness, but merely that the protocol can sometimes run to completion. They imply basic properties, such as that message formats are compatible from one step to the next. Though logically trivial,

their machine proofs require significant effort (or computation) due to the complexity of the terms that arise and the number of choices that can be made at each step. I proved cases corresponding to runs with up to three agents plus the server and spy. General theorems for n agents could be proved by induction on n, but the necessary effort hardly seems justified.

Security properties are proved, as always, by induction over the protocol definition. For this protocol, the main inductive set (recur) is defined in terms of another (respond). All but the most trivial proofs require induction over both definitions. The inner induction over respond might be proved as a preliminary lemma if it is used more than once or if its proof is complicated.

The set responses specifies the general form of the outputs generated by respond:

```
(PA,RB,K): respond evs ==> RB: responses evs
```

This easily-proved result justifies performing induction over responses instead of respond; if it leads to a proof at all, it will lead to a simpler proof.

Elementary results are no harder to prove than for a fixed-length protocol. The outer induction yields six subgoals: one for each protocol step, plus the base and fake cases. The inner induction replaces the step 3 case by two subcases, namely the server's base case and inductive step. There are thus seven subgoals, few of which typically survive simplification. Only the theorems described below have difficult proofs.

Nonces generated in requests are unique. This theorem states that there can be at most one hashed value containing the key of an uncompromised agent ($A \notin lost$) and any specified nonce value, Na.[2]

```
∃ B' P'. ∀ B P.
  Hash (|Key(shrK A), Agent A, Agent B, Na, P|)
  ∈ parts (sees lost Spy evs)
  ⟶  B=B' ∧ P=P'
```

Although it is not used in later proofs, unicity of nonces is important. It lets agents identify runs by their nonces. The theorem applies to all requests, whether generated in step 1 or step 2. For the Otway-Rees protocol, each of the two steps requires its own theorem. The reasoning here is similar, but one theorem does the work of two, thanks to the protocol's symmetry. The nesting of requests does not affect the reasoning.

Secrecy proofs require the lemma

```
(Key K ∈ analz (ins(Key Kab)(sees lost Spy evs)))
=
(K=Kab ∨ Key K ∈ analz (sees lost Spy evs))
```

---

[2] The set sees *lost* Spy *evs* consists of everything the spy can see: all the messages in the trace *evs* and the long-term keys of all the agents in the set *lost*. This particular theorem is not concerned with the spy but with possible message histories.

89

where Kab is a session key. (Equality between formulae denotes if-and-only-if.) Rewriting using it extracts session keys from the argument of analz. The lemma may be paraphrased as saying that existing session keys cannot be used to learn new ones [9]. It is hard to prove. For the induction to go through, it must be generalized to an arbitrary set of session keys:

```
∀ K KK. KK ⊆ Compl (range shrK) ⟶
  (Key K ∈ analz (Key''KK ∪ (sees lost Spy evs)))
  =
  (K ∈ KK ∨ Key K ∈ analz (sees lost Spy evs))
```

The inner induction over respond leads to excessive case splits. It was to simplify this proof that I defined the set responses.

Unicity for session keys is unusually complicated because each key appears in two certificates. Moreover, the certificates are created in different iterations of respond. The unicity theorem states that, for any K, if there is a certificate of the form

$$\mathsf{Crypt}(\mathsf{shrK}\,A)\{K, B, Na\}$$

(where A and B are uncompromised) then the only other certificate containing K must have the form

$$\mathsf{Crypt}(\mathsf{shrK}\,B)\{K, A, Nb\},$$

for some Nb. If $(PB, RB, K) \in \mathsf{respond}\,evs$ then

```
∃ A' B'. ∀ A B N.
  Crypt (shrK A) {|Key K,Agent B,N|} ∈ parts(RB)
  ⟶  (A'=A ∧ B'=B) ∨ (A'=B ∧ B'=A)
```

This theorem seems quite strong. An agent who receives a certificate immediately learns which other agent can receive its mate, subject to the security of both agents' long-term keys. One might hope that security of session keys would follow without further ado. Informally, we might argue that the only messages containing session keys contain them as part of such certificates, and thus the keys are safe from the spy. But such reasoning amounts to another induction over all possible messages in the protocol. The theorem must be stated (stipulating A, $A' \notin lost$) and proved:

```
Crypt (shrK A) {|Key K, Agent A', N|}
  ∈ parts (sees lost Spy evs)
  ⟶ Key K ∉ analz (sees lost Spy evs)
```

The induction is largely straightforward except for the step 3 case. The inner induction over respond leads to such complications that it must be proved beforehand as a lemma. If $(PB, RB, Kab) \in \mathsf{respond}\,evs$ then

```
∀ A A' N. A ∉ lost ∧ A' ∉ lost
  ⟶
  Crypt (shrK A) {|Key K,Agent A',N|} ∈ parts(RB)
  ⟶
  Key K ∉ analz (ins RB (sees lost Spy evs))
```

A slightly stronger result is that the key enclosed in a certificate does not reach any unintended agents, even honest ones.

Although each session key appears in two certificates, they both have the same format. A single set of proofs applies to all certificates. The protocol's symmetry halves the effort compared with Otway-Rees, which requires two sets of proofs.

7. Potential Attacks

One must never assume that a "verified" protocol is infallible. The proofs are subject to the assumptions implicit in the model. Attacks against the protocol or implementations of it can still be expected. One "attack" is quite obvious: in step 2, agent B does not know whether A's message is recent; at the conclusion of the run, B still has no evidence that A is present. The spy can masquerade as A by replaying an old message of hers, but cannot read the resulting certificate without her long-term key.

Allowing type confusion (such as passing a nonce as a key) often admits attacks [5, 7] in which one encrypted message is mistaken for another one that is intended for another purpose. The recursive authentication protocol appears to be safe from such attacks: it has only one form of encrypted message, with only one purpose. However, the implementation of encryption must be secure.

The original protocol description suggested an unsound form of encryption. Each session key was encrypted by forming its exclusive "or" with a hash value, used as a one-time pad. Unfortunately, each hash value was used twice: B's session keys Kab and Kbc were encrypted as

$$Kab \oplus \mathsf{Hash}\{Kb, Nb\} \quad \text{and} \quad Kbc \oplus \mathsf{Hash}\{Kb, Nb\}.$$

By forming their exclusive "or", an eavesdropper could immediately obtain $Kab \oplus Kbc$, $Kbc \oplus Kcd$, etc. Compromise of any one session key would allow all the others to be read off. Using say $\mathsf{Hash}\{Kb, Nb+1\}$ to encrypt the second key would be secure, assuming a good hash function.

This attack (found by Peter Ryan and Steve Schneider) is a valuable reminder of the limitations of formal proofs. It does not contradict the proofs, which regard encryption as primitive. The inductive approach can probably be extended to cope with exclusive "or". But such a low-level description violates the principle of *separation of concerns*. It requires longer proofs and can only be recommended if the protocol specifically requires one form of encryption. Perhaps we need formal tools to allow implementations of encryption to be proved correct independently of the protocols using them.

8. Conclusions

Analyzing this protocol took about two weeks of normal working days, including the formalization of hashing and experimentation with different formats for certificates. Streamlining and maintaining the proofs has taken additional time. The proofs are modest in scale: fewer than 30 results are proved, using under 135 tactic commands; they run in less than five minutes.

The inductive approach readily models this complex protocol. It is perfectly suited to Isabelle/HOL's inductive definition package, simplifier and classical reasoner. Proofs can be generated using modest human and computational resources. The approach does not search for attacks, but establishes positive guarantees. In the present case, it has suggested ways of strengthening the protocol by adding explicitness. Bull has suggested that the protocol might be modified to distribute session keys between agents that are not adjacent in the request chain. Such variants can probably be analyzed with little difficulty by modifying the existing proof script.

References

[1] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.

[2] Dominique Bolignano. An approach to the formal verification of cryptographic protocols. In *Third ACM Conference on Computer and Communications Security*, pages 106–118. ACM Press, 1996.

[3] John A. Bull and David J. Otway. The authentication protocol. Technical Report DRA/CIS3/PROJ/CORBA/SC/1/CSM/436-04/03, Defence Research Agency, Malvern, UK, 1997.

[4] M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. *Proceedings of the Royal Society of London*, 426:233–271, 1989.

[5] Gavin Lowe. Casper: A compiler for the analysis of security protocols. These proceedings.

[6] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems. Second International Workshop, TACAS '96*, LNCS 1055, pages 147–166, 1996.

[7] Catherine A. Meadows. Analyzing the Needham-Schroeder public-key protocol: A comparison of two approaches. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *Computer Security — ESORICS 96*, LNCS 1146, pages 351–364. Springer, 1996.

[8] Dave Otway and Owen Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, January 1987.

[9] Lawrence C. Paulson. Proving properties of security protocols by induction. These proceedings.

[10] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*. Springer, 1994. LNCS 828.

A Isabelle Specifications and Theorems

Isabelle users will understand this material best, but I hope all readers will gain an impression of the notation. Figure 2 defines the relation respond evs, which specifies the server's interpretation of requests. The set responses evs is omitted because it does not form part of the specification itself (it merely simplifies some of the proofs). Figure 3 specifies the protocol itself. I have omitted some comments.

Figure 4 presents the formal statements of the main theorems of §6., in close to raw Isabelle syntax. The proof scripts are omitted because they are unintelligible. A proof requires four commands on average, of which at least two are quite predictable: induction and simplification.

```
consts       respond :: "event list => (msg*msg*key)set"
inductive "respond evs"
  intrs
    (*The message "Agent Server" marks the end of a list.*)
    One  "[| A ≠ Server;  Key KAB ∉ used evs |]
         ⟹ (Hash[Key(shrK A)]
                        {|Agent A, Agent B, Nonce NA, Agent Server|},
              {|Crypt(shrK A){|Key KAB,Agent B,Nonce NA|}, Agent Server|},
              KAB)   ∈ respond evs"

    (*The most recent session key is passed up to the caller*)
    Cons "[| (PA, RA, KAB) ∈ respond evs;
             Key KBC ∉ used evs;  Key KBC ∉ parts {RA};
             PA = Hash[Key(shrK A)] {|Agent A, Agent B, Nonce NA, P|};
             B ≠ Server |]
         ⟹ (Hash[Key(shrK B)] {|Agent B, Agent C, Nonce NB, PA|},
              {|Crypt (shrK B) {|Key KBC, Agent C, Nonce NB|},
                Crypt (shrK B) {|Key KAB, Agent A, Nonce NB|},
                RA|},
              KBC)    ∈ respond evs"
```

Figure 2. Specifying the Server

```
consts        recur    :: agent set => event list set
inductive "recur lost"
  intrs
        (*Initial trace is empty*)
    Nil  "[] ∈ recur lost"

        (*The spy MAY say anything he CAN say.*)
    Fake "[| evs ∈ recur lost;  B≠Spy;
            X ∈ synth (analz (sees lost Spy evs)) |]
         ⟹ Says Spy B X  # evs ∈ recur lost"

        (*Alice initiates a protocol run.*)
    RA1  "[| evs ∈ recur lost;  A≠B;  A≠Server;  Nonce NA ∉ used evs |]
         ⟹ Says A B
              (Hash[Key(shrK A)]
                {|Agent A, Agent B, Nonce NA, Agent Server|})
            # evs ∈ recur lost"

        (*Bob's response to Alice's message.  C might be the Server.*)
    RA2  "[| evs ∈ recur lost;  B≠C;  B≠Server;  Nonce NB ∉ used evs;
            Says A' B PA ∈ set_of_list evs;
            PA = {|XA, Agent A, Agent B, Nonce NA, P|} |]
         ⟹ Says B C (Hash[Key(shrK B)]{|Agent B, Agent C, Nonce NB, PA|})
            # evs ∈ recur lost"

        (*The Server receives Bob's message and prepares a response.*)
    RA3  "[| evs ∈ recur lost;  B≠Server;
            Says B' Server PB ∈ set_of_list evs;
            (PB,RB,K) ∈ respond evs |]
         ⟹ Says Server B RB # evs ∈ recur lost"

        (*Bob receives the returned message and compares the Nonces with
          those in the message he previously sent the Server.*)
    RA4  "[| evs ∈ recur lost;  A≠B;
            Says C' B {|Crypt (shrK B) {|Key KBC, Agent C, Nonce NB|},
                        Crypt (shrK B) {|Key KAB, Agent A, Nonce NB|},
                        RA|}
              ∈ set_of_list evs;
            Says B  C {|XH, Agent B, Agent C, Nonce NB,
                        XA, Agent A, Agent B, Nonce NA, P|}
              ∈ set_of_list evs |]
         ⟹ Says B A RA # evs ∈ recur lost"
```

Figure 3. Specifying the Protocol

```
(PA,RB,KAB) ∈ respond evs ⟹ RB ∈ responses evs


[| evs ∈ recur lost; A ∉ lost |]
⟹ ∃ B' P'. ∀ B P.
        Hash {|Key(shrK A), Agent A, B, NA, P|}
           ∈ parts (sees lost Spy evs)   ⟶   B=B' ∧ P=P'


evs ∈ recur lost ⟹
∀ K KK. KK <= Compl (range shrK) ⟶
         (Key K ∈ analz (Key''KK Un (sees lost Spy evs))) =
         (K ∈ KK ∨ Key K ∈ analz (sees lost Spy evs))


[| evs ∈ recur lost;  KAB ∉ range shrK |] ⟹
         Key K ∈ analz (ins (Key KAB) (sees lost Spy evs)) =
         (K = KAB ∨ Key K ∈ analz (sees lost Spy evs))


(PB,RB,KXY) ∈ respond evs
⟹ ∃ A' B'. ∀ A B N.
              Crypt (shrK A) {|Key K, Agent B, N|} ∈ parts {RB}
              ⟶    (A'=A ∧ B'=B) ∨ (A'=B ∧ B'=A)


[| (PB,RB,KAB) ∈ respond evs;  evs ∈ recur lost |]
⟹ ∀ A A' N. A ∉ lost ∧ A' ∉ lost ⟶
              Crypt (shrK A) {|Key K, Agent A', N|} ∈ parts{RB} ⟶
              Key K ∉ analz (ins RB (sees lost Spy evs))


[| Crypt (shrK A) {|Key K, Agent A', N|} ∈ parts (sees lost Spy evs);
     A ∉ lost;  A' ∉ lost;  evs ∈ recur lost |]
⟹ Key K ∉ analz (sees lost Spy evs)
```

Figure 4. Some Theorems in Isabelle's Syntax

Using One-Way Functions for Authentication

Li Gong

University of Cambridge Computer Laboratory
Cambridge CB2 3QG, England
lg@cl.cam.ac.uk

July 1989

Abstract- Techniques are suggested to construct authentication protocols on a basis of one-way functions rather than encryption algorithms. This approach is thought of interest for several reasons. It appears that this approach could achieve, at least, equally simple and capable protocols.

1 Introduction

In computer networks or distributed computing systems it is necessary to have procedures, known as authentication protocols, by which pairs of principles satisfy themselves mutually about each other's identity. Most if not all existing such protocols are based on similar principles to those in the often cited work of Needham and Schroeder [5]. The most common approach of authentication is by means of shared secrets, usually encryption keys. A typical protocol normally makes use of some cryptographic algorithms, either private-key or public-key cryptography, for encryptions and decryptions [5,1].

In this paper we provide a framework by which authentication protocols can be constructed on a basis of one-way functions, rather than encryption algorithms. A function f is a one-way function if, for any argument x in the domain of f, it is easy to compute $f(x)$, yet, for almost all y in the range of f, it is computationally infeasible to solve the equation $y = f(x)$ for any suitable argument x [2].

This approach is of interest in that compared with encryption algorithms, one-way functions may be easier to design because there is no need to make them invertible. Also, since the primary capability of a one-way function is to provide an integrity

service but not a confidentiality service, export control of such products to foreign vendors or customers may be less restricted.

2 A Mutual Authentication Protocol

Using the conventional notations, we describe the protocol and explain it in a chronological order of beliefs held, computations done, messages sent, and the evolution of beliefs as results of such actions. The notation ',' represents concatenation. f, g are both one-way functions. They may be identical or variations of the same one-way function.

Message 1. $A \rightarrow B$: A, B, na
Message 2. $B \rightarrow S$: A, B, na, nb
Message 3. $S \rightarrow B$: ns, f(ns, nb, A, Pb)$\oplus$(k, ha, hb), g(k, ha, hb, Pb)
Message 4. $B \rightarrow A$: ns, hb
Message 5. $A \rightarrow B$: ha

Initially at the start of a transaction, clients Alice and Bob each shares a secret with the authentication server Sue, most probably in the form of passwords Pa and Pb. Alice initiates the transaction and generates a nonce na. A nonce is a number which has not been used before for the same purpose.

Message 1. $A \rightarrow B$: A, B, na

Bob responses by generating nonce nb and passing on the request to Sue.

Message 2. $B \rightarrow S$: A, B, na, nb

When Sue receives this request, she generates a nonce ns and computes $(k, ha, hb) = f(ns, na, B, Pa)$. Here k is a secret to be shared between the two clients. ha and hb are called handshake numbers. Note that the value of k is dependent partially on the choice of ns. Now she computes $f(ns, nb, A, Pb)$ and g to construct message 3.

Message 3. $S \rightarrow B$: ns, f(ns, nb, A, Pb)$\oplus$(k, ha, hb), g(k, ha, hb, Pb)

Bob computes $f(ns, nb, A, Pb)$ to retrieve (k, ha, hb) from the second item, and computes g to check against the third item that they have not been tampered with. Satisfied, he knows that he has correctly received k which must be originally generated by Sue, because the computation requires the use of Pb which he believes known only to him and Sue. He also knows that k is fresh because the computation requires nonce nb. This defeats a replay attack. Identifiers A and B are also necessary to defeat an attack in which an intruder impersonates Alice (or Bob)

without being detected by the other client. Then Bob forwards ns and sends hb. There is no need to directly use k to acknowledge its receipt, because it can only be correctly received together with ha and hb.

Message 4. $B \rightarrow A$: ns, hb

Now Alice computes $f(ns, na, B, Pa)$ to get (k, ha, hb). If her hb matches the one sent by Bob, she knows he knows he has correctly received a fresh secret k, and thus she deduces that she has also received k correctly. Then she acknowledges this in message 5.

Message 5. $A \rightarrow B$: ha

Comparing this number with the ha he computes earlier, Bob knows Alice has correctly received k and she knows he too has received it. Now the transaction terminates successfully in a state that both clients know they share with each other a fresh secret, and they both know that the other client knows about this [1].

3 Remarks

First of all, if k is to be used later as an encryption key, the process of selecting ns may be reiterated until a good key is found. However, if k has to be generated separately, or if k partially contains some structured or predetermined data, by nature of one-way functions it is computationally infeasible to select a suitable ns. In this case, a simple solution is to generate k and three nonces ns, ha, and hb, and modify message 3 and 4 as follows,

Message 3. $S \rightarrow B$: ns, f(ns, na, B, Pa)$\oplus$(k, ha, hb), g(k, ha, hb, Pa),
                f(ns, nb, A, Pb)$\oplus$(k, ha, hb), g(k, ha, hb, Pb)
Message 4. $B \rightarrow A$: ns, f(ns, na, B, Pa)$\oplus$(k, ha, hb), g(k, ha, hb, Pa), hb

The particular form that one-way functions are used in the protocol is for example $f(ns, nb, A, Pb)$. Since all but one input item, Pb in this case, is available to anyone who monitors the network traffic, it is required that an intruder be not able to find out Pb by directly inverting f.

Moreover, it would appear that an intruder could search the password space to reveal Pb. Here we see that ns makes a dictionary attack much more difficult. We also refer you to [3] for further discussion on dealing with poorly chosen passwords.

Finally we point out that adopting the new approach to construct authentication protocols on the basis of one-way functions would not necessarily reduce the capability of authentication protocols. For example, it is conceivable that a Kerberos

[4] like service can be similarly designed. Of course if timestamps are used, as in Kerberos, four messages will be sufficient. Protocols of various other forms can also be derived.

4 Acknowledgment

The effort to replace encryptions with one-way functions in authentication protocols was triggered in March partially by a question from Roger Needham. The fact that structurally very simple authentication protocols can be built on the basis of one-way functions was fully realized when the author had a discussion with Martín Abadi and Mike Burrows at DEC SRC in May. Mark Lomas, Roger Needham, David Wheeler and Raphael Yahalom gave helpful comments on an earlier version of the paper. This work also benefits from discussion with Jerry Saltzer of MIT.

References

[1] M. Burrows, M. Abadi, and R.M. Needham, "A Logic of Authentication", DEC System Research Center Technical Report No.39, February, 1989.

[2] W. Diffie and M.E. Hellman, "New Directions in Cryptography", *IEEE Transactions on Information Theory*, Vol. IT-22, No.6, November, 1976, pp.644-654.

[3] T.M.A. Lomas, L. Gong, J.H. Saltzer, and R.M. Needham, "Reducing Risks from Poorly Chosen Keys", to appear in Proceedings of the 12th ACM symposium on Operating Systems Principles, December, 1989.

[4] S.P. Millen, C. Neuman, J.I. Schiller, and J.H.Saltzer, "Kerberos Authentication and Authorization System", *Project Athena Technical Plan*, Section E.2.1, MIT, July, 1987.

[5] R.M. Needham and M.D. Schroeder, "Using Encryptions for Authentication in Large Networks of Computers", *Communication of the ACM*, Vol.21, No.12, December, 1978, pp.993-999.

internet.com **(Wĕbopēdia)** The #1 online encyclopedia dedicated to computer technology

Enter a keyword...

Go!

...or choose a category.

All Categories ▾ Go!

MENU

Home
Term of the Day
New Terms
New Links
Quick Reference
Search Tool
Link to Us
Advertising

Talk To Us...

Internet Jobs
Tech Support
Submit a URL
Request a Term
Report an Error

internet.com

Internet News
Internet Stocks/VC
Internet Technology
Windows Internet Tech.
Linux/Open Source
Web Developer
E-Commerce/Marketing
ISP Resources
Downloads
Internet Resources
Internet Lists
International

Search Internet.com
Advertising Info
Corporate Info
Internet Trade Shows

internet commerce

Small Business
Address Verification
Build Sticky eStores
Advertise a Coupon
Promotional Products
Accept Credit Cards
Download Solutions
Create a Web Store
fax & vmail for free

Windows
101 BEST BUSINESS SITES

Network Neighborhood

Last modified: October 12, 1997

A Windows 95 folder that lists computers, printers and other resources connected to your local-area network (LAN). By default, a Network Neighborhood icon appears on your desktop, and the folder is also accessible from within the Windows 95 Explorer. The Network Neighborhood is designed to replace the *drive mapping* older system, which associates a letter with each shared disk drive. Many programs, however, still require drive mapping.

The Network Neighborhood serves no purpose if your computer is not connected to a LAN, except that it is required to link two computers using Windows 95's Direct Cable Connection (DCC) feature.

Related Categories

Windows

Related Terms

DCC

local-area network

Windows 95

(Wĕbopēdia)
Give Us Your Feedback

> For internet.com pages about **Network Neighborhood**, *CLICK HERE*. Also, check out the following links!

LINKS
⌇⌇ = Great Page!

Sorry, no links for this term yet.

- **Interesting Articles from Today on internet.com:**

HP, IBM Developing Wearable Net Devices
Two blue chip technology companies, IBM and Hewlett-Packard, are talking about releasing wearable Net devices.

HAHTsite: A Flexible Apps. Server
With an outstanding and flexible development environment, HAHTsite from Haht Software is a feature-rich product.

BEST AVAILABLE COPY

MIT Profs Sue Ask Jeeves
Here's a question to ask the online search engine Ask Jeeves: Did your creators
steal patented technology when they developed you?

Win95 Tutorial

My Computer

Network Neighborhood

Recycle Bin

An Introduction to using Windows 95 at Valencia Community College

note: this tutorial looks best with the window maximized see how to do it.

How to use this tutorial

Index of Topics

Win95 Tutorial
Shortcut to Intro

Start

Faculty Resource Center

- What is the desktop? How do I use it?
- Finding files in My Computer
- Finding files with Windows Explorer
- Using Network Neighborhood
- Maximizing and minimizing windows
- Adjusting the size of windows
- Using the taskbar
- Adding shortcuts to Win95
- Using the Start button
- Using the Recycle Bin
- Shutting down Win95
- How do I move the taskbar back to the bottom of the screen?
- The taskbar has disappeared, how do I get it back?
- How do I change the background image in Win95?

Network Neighborhood

My Computer

Network Neighborhood

Recycle Bin

Win95 Tutorial
Shortcut to Intro

Start

This icon will appear on your desktop if you are connected to a network of computers. When you left click on the Network Neighborhood icon you will see a menu listing the other computers attached to yours. Putting computers on a network allows you to share resources such as documents or programs between different computers. Here is an example of the network at the Faculty Resource Center

Netwo... File Edit View Help

- Entire Network
- Boo
- Bruno
- Foo
- Infectious
- Library
- Oops
- Orbity

13 object(s)

Faculty Resource Center
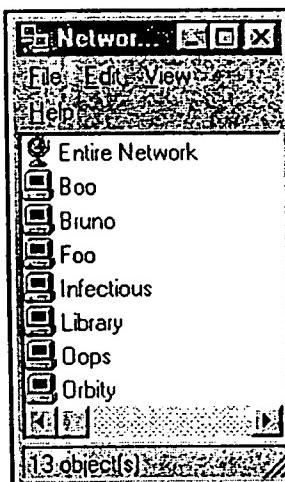
Each terminal icon with a name next to it represents a different computer on the network. Every machine on the network has it's own unique name, for example Boo, Bruno and Foo are separate PC's running in our lab. If you left click the icon's in network neighborhood you will see which drives are shared.

Oops

- d
- share_c
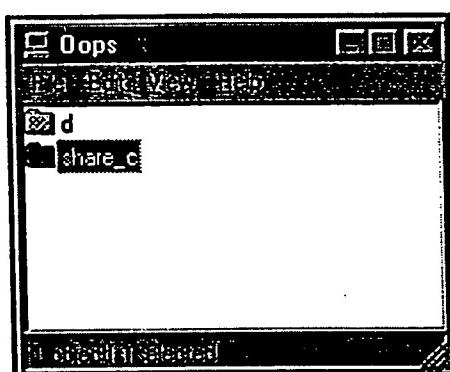
When a drive is *shared* it means that other computers on the network can see the contents of that drive. In this example the C hard drive has been shared so we can go in and look at files on Oops, we can even run some programs that are on Oops even if we are on a different computer in the lab.

Security first: Neil Stewart of Aspects Software   Ashley Coombes

INFORMATION TECHNOLOGY M-COMMERCE

How to hold mobile users to account

Mark Nicholson
on a development

card, which was initially
designed for the cellphone
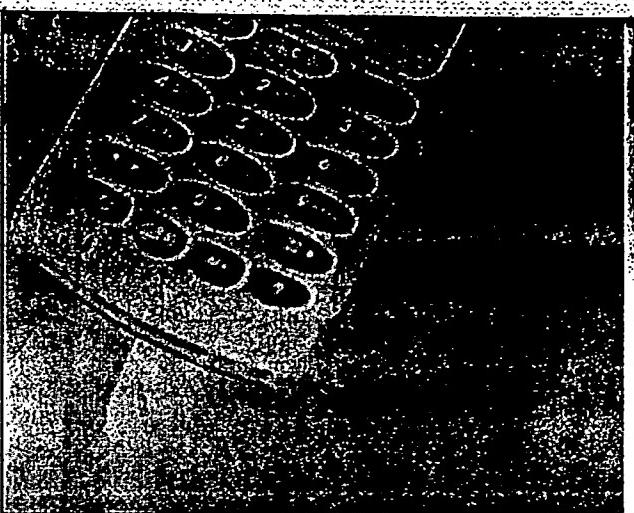
Security first: Neil Stewart of Aspects Software   Ashley Coombes

INFORMATION TECHNOLOGY, M-COMMERCE

How to hold mobile users to account

Mark Nicholson on a development of the Sim card that could help banks retain customers

Forget branches. Move even more from telephone banking into internet banking. The future of banking may lie in the Sim card, the little plastic insert in your mobile phone which identifies you as its owner, and ties you to your cellular network.

That is the contention of Aspects Software, an Edinburgh company that has designed an innovative amalgam of the Sim card and high-security transaction applications enabling safe financial transactions to be made over a cellphone.

On one level therefore its SecureSim product is another step towards m-commerce. With the Sim for mobile, its founders believe, however, that its distinctive feature is not so much what it offers the consumer as what it may offer the banks — branding and customer ownership.

card, which was initially designed for the cellphone operators to offer users security, but also to peg them to the operators' network. Mr Stewart describes the Sim card as "a piece of real estate" within the mobile phone — one over which banks could take ownership through a joint agreement with a cellphone operator.

For example, banks could embed their services into the Sim in a way that would not only allow bank customers to do their transactions by phone, but also show the bank's name or logo on the phone display — keeping the bank's brand firmly in sight.

Aspects, which specialises in GSM and smart card technology, diagnostics and prototyping, has brought together the Sim and a transactions application called Multos, a "virtual machine" which has the highest level of security certification. Through its use, SecureSim can allow credit, debit, Visa or Mastercard schemes or purse schemes for payments.

Mr Stewart sees this combination of security, an emerging standard with which Wap systems are still wrestling, and brand identification as potentially compelling for the biggest challenge facing most retail banks, and also for big

your mobile phone which identifies you as its owner and ties you to your cellular network.

That is the contention of Aspects Software, an Edinburgh company that has designed an innovative amalgam of the Sim card and high security transaction applications, enabling safe financial transactions to be made over a cellphone.

On one level, therefore, its SecureSim product is another step towards m-commerce, with the "m" for mobile. Its founders believe, however, that its distinctive feature is not so much what it offers the consumer, as what it may offer the banks - branding and customer ownership.

Peter Burt, chief executive of Bank of Scotland, mused recently that the biggest challenge facing most retail banks was "turning ourselves into telecommunications companies". He pointed out that already 80 per cent of banking transactions take place outside the bank, mostly over the telephone, and that the proportion will rise.

Like many other banks, BoS has begun trialling wireless application protocol (Wap) systems, which allow customers access to their bank accounts via the new generation of internet-linked mobiles.

The trouble for the banks, however, is that using Wap technology eventually risks giving away "ownership" of the customer to the network operators. According to Neil Stewart, chief executive of Aspects, the advent of Wap could lead to companies such as Virgin, which is already turning itself into a network operator with its branded cellular service, offering almost the full range of banking products, and leaving the banks totally cut out of the value transaction loop. What price an eventual Vodafone Bank, he asks.

Behind the development of Aspects' Sim technology lies a possible solution for the banks. If telcos can become banks, why can't banks become telcos? he asks, echoing Mr Burt's thoughts. The solution, as he sees it, is for the banks as "customer owners" to become "content providers" and to take control of the network through which their customers transact with the bank.

The trick lies in the Sim

Sim in a way that would not only allow bank customers to do their transactions by phone, but also show the bank's name or logo on the phone display - keeping the bank's brand firmly in sight.

Aspects, which specialises in GSM and smart card technology, diagnostics and prototyping, has brought together the Sim and a transactions application called Multos, a "virtual machine" which has the highest level of security certification. Through its use, SecureSim can allow credit, debit, Visa or Mastercard schemes, or e-purse schemes for payments.

Mr Stewart sees this combination of security - an issue with which Wap systems are still wrestling - and brand identification as potentially compelling for banks, and also for big retailers. It could, for example, supersede "smart cards" mainly because it requires far less point-of-sale infrastructure for the retailers - the main impediment so far to the widespread use of smart cards even in the US.

To back the SecureSim technology, Aspects (www.aspects-sw.com) has created a separate company, mobEcom, which recently received £1m in venture capital backing from Scottish Equity Partners, Scotland's second biggest technology investor. Cellular network operators are already showing interest, along with at least one big bank.

Mr Stewart believes more venture funding will be forthcoming for mobEcom, which he also says will be first to market with the new Sim-secure transactions technology.

"No one else has yet made a secure payment environment which is really properly secured phone that the banks can say, 'I'll run with this'," he says. "We're saying to the banks, here's the base, put onto it whatever you want, and from this secure environment, we'll put that range of applications into wireless space."

Aspects is not the only company active in this field, though. Finland's Sonera SmartTrust, for example, is pioneering a technique through which public key infrastructure (PKI) security functions can be embedded into a mobile phone Sim, turning the phone into the electronic equivalent of a passport.

as many computer managers in the UK want to see Microsoft broken up as want to see it left alone.

This is the crushing finding of this week's Computer Weekly/Harvey Nash Big Question poll.

Forcing Microsoft to make Windows an open-source product and restricting its business practices are also twice as popular as leaving the company alone.

Readers were responding to the recent US Court judg-

One Computer Weekly reader called for Microsoft to be barred from the Internet altogether, "Microsoft's dominance of the desktop has been beneficial but this is a battle to stop Microsoft stifling the Web," he said.

Another said a breakup was necessary to protect the marketplace.

There was concern that open source and business restrictions would not benefit the consumer. Several readers said open source could cause problems with a lack of standards in the

considered its competitors a worse alternative. "If forcing innovation means providing greater opportunity for the likes of Larry Ellison to force technologies such as the network computer onto us, then I say, stay as we are."

Others said Microsoft would split into several companies, regardless of the court's decision.

"The speed of innovation is such that if Microsoft wishes to compete over the long term, it will need to split into relatively smaller, more agile, focused units," a reader said.

The big question

Which remedy would you prefer in the Microsoft case?
a) Split Microsoft into consumer. operating system and application companies?
b) Force the company to make Windows open source. allowing users to develop and share extensions to the operating system?
c) Curb acqusitive business practices and those that stifle technological innovation?

None — Split Microsoft — 12% — Restrict business — 26% — Windows open source

---
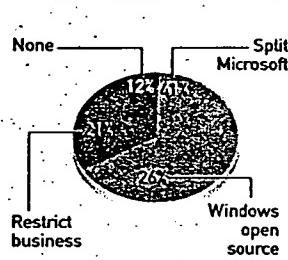
Handset giants launch security system

Tony Savvas

THREE major mobile handset manufacturers have formed a mobile electronic transaction initiative to standardise secure mobile commerce.

Nokia, Ericsson and Motorola say the initiative is designed to deliver integrity confidentiality, authentication and non-repudiation to transactions completed using mobile handsets.

They have promised that their open standards approach build around Wim (wireless identity model) will deliver this breakthrough by 2001. They rejected suggestions that current mobile

handsets offered poor security and maintained that Wap handsets already in the market offered at least as good security as that offered by the Internet.

However, mobile phones are increasingly being used for large-value transactions in the banking world. The consortium wants to ensure that their

phones will offer encryption techniques that are as powerful as those used by the banks.

Basic requisites of the framework being offered will cover digital certificates, server storage, server communications with transport mechanisms including Wap and short-wave radio technology Bluetooth.

Working together. Top trio Nokia. Ericsson and Motorola promise secure mobile commerce

---
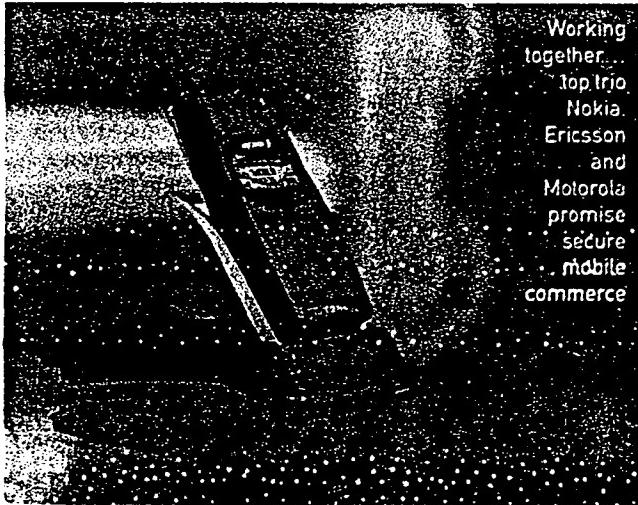
The outs

Mike Simons

LAMBETH Council is lock in a housing benefits contr with outsourcer Capi despite a 40,000 backlog claims and a series of fai rectification programmes.

Resource-strapped La beth is spending up to £1. to take back managem control of the service.

But the council has ru out the option of exercis its legal right to make Cap pick up the cost of the mo

The council's execut director of finance Mich Crich told Computer Wee that such a move could aff Capita's work on the rest the contract, which cov council tax, call centre cashier services.

Lambeth's plight follo last month's admission Treasury Minister Dawn marolo that the Governm would not demand comp sation from Andersen for troubled NIRS2 natio insurance records contr for fear of damaging fut relationships.

The two cases illustrate dangers that could fa organisations, public and vate, which outsource c business processes. WI problems develop, clients find themselves at the me of their outsourcer, howe strong their contracts.

Lambeth admitted reliance on Capita in a rep presented to the counc Policy Committee on 3 Ap It said, "The council is leg in a position to suspend benefits service from the c tract and to provide or p cure the service fr elsewhere at Capit expense.

However, the key questi

sis'imperfecta. The mutant initially caught her attention with its small size. When she x-rayed it, she found a mass of broken ribs. On closer inspection of the mutant strain, she found that it forms cartilage normally during early development, but its bones in adulthood are fragile. Although she has not identified the exact mutation at fault, it seems to be near a collagen-encoding gene fingered in human cases of osteogenesis imperfecta. Fisher suspects that this mutant strain could prove valuable for research into collagen's role in bone formation and maintenance.

Whereas broken bones are relatively easy to spot, problems in biochemistry can be harder to detect, even in the see-through zebrafish. But here, too, scientists are hoping the animal will help answer some difficult questions. Steven Farber of Thomas Jefferson University in Philadelphia and Michael Pack of the University of Pennsylvania School of Medicine in Philadelphia and their colleagues have devised a way to observe the biochemical reactions of digestion in living zebrafish. To identify genes that regulate one part of digestion, lipid processing—known to influence

the development of colon cancer, heart disease, and other human ills—the team has designed lipid molecules that glow when they are digested by a key enzyme in the intestine. When the scientists feed this molecule to zebrafish larvae, they can see the molecule light up in the digestive tract and liver and then
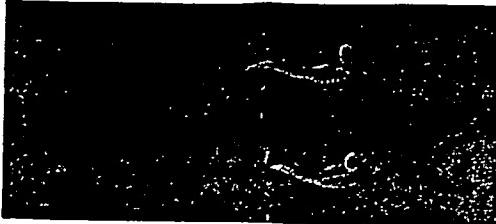
Healthy glow. A custom-made lipid, designed to identify fish with faulty digestion, lights up in the digestive tract of zebrafish larvae.

travel to the gallbladder. Although the screen is in its early stages and the scientists have only begun to identify potential mutants, developmental biologist Didier Stainier of the University of California, San Francisco, is impressed. "If you can actually ask questions

about how these intestinal cells are processing a substrate, that's very powerful," he says. The team hopes to design other molecules to probe the digestion of carbohydrates and other molecules. "We're visualizing biochemical processes in living vertebrates," says Farber. "Zebrafish is the only game in town where you can do that."

The most difficult part of the process is still tracking down the mutant gene itself, but scientists say that ongoing work in zebrafish genomics is making that task easier. And the likely launch of an effort to sequence the zebrafish genome (*Science*, 5 May, p. 787) will also ease that task. Genome projects in the mouse and human, says developmental biologist Nancy Hopkins of the Massachusetts Institute of Technology, will only make the zebrafish more important. Those projects will turn up thousands of unknown genes, she says, and it is likely to be easier to figure out what they do in the fish. Says Hopkins: "We've barely begun to tap" the potential of zebrafish. —GRETCHEN VOGEL

DIGITAL ENCRYPTION

Algorithmic Gladiators Vie For Digital Glory

As NIST zeroes in on a new cryptographic standard, the competitors scramble to face an unforeseen threat—from lawyers

And then there were five. For 2 years, glory-seeking cryptographers from across the globe have been cracking one another's ciphers, trying to establish their own algorithms as the new standard in encryption. In mid-April the five finalist teams faced off in New York. They subjected each other's algorithms to the withering fire of cryptographic attack after cryptographic attack, while judges observed the melee. But as the smoke cleared, the contestants found themselves facing a menace from a new and unexpected quarter: the realm of patent law.

The five finalist algorithms—MARS, Twofish, Rijndael, RC6, and Serpent—are vying to be the new standard in encryption, replacing the aging Digital Encryption Standard (DES), endorsed by the National Bureau of Standards in the mid-'70s. Thanks to the government's stamp of approval, DES has become perhaps the most widely used encryption system in the world. The new algorithm, selected by the National Institute of Standards and Technology (NIST)—the Bureau of Standards' successor—will replace DES and should assume its mantle of preeminence. No money is at stake in the competition; under NIST's licensing terms,

the inventor of Advanced Encryption Standard (AES) will not benefit financially. "The big thing, personally, is the fun of doing it," says John Kelsey of Counterpane Internet Security in San Jose, California. "If you're in block ciphers, it's the coolest thing you can do, as far as I can tell."

Outside the arena, however, the stakes are serious indeed. If someone were to crack the AES a few years down the line, all the reams of data encrypted with NIST's standard could be compromised. Medical records, bank transactions, and other confidential information would potentially be wide open to anyone with the know-how, and it would take years for engineers to replace the cracked algorithm in smart cards, computers, and descrambler boxes.

Fears of such a breach are what drove officials to seek a replacement for DES in the first place. DES was designed to take a stream of digital data, split it into 64-bit chunks, and encipher it. In theory, an eavesdropper could not decipher the data without guessing the 56-bit cryptographic "key" that opens the cipher—a secret shared by only the sender and intended receiver.

By the early 1990s, fissures had begun to

show in DES's security. Cryptographers such as Eli Biham and Adi Shamir of the Technion–Israel Institute of Technology in Haifa developed new attacks such as "differential" cryptanalysis, in which a cryptographer tries to crack an algorithm by feeding very slightly different data into it and comparing how the encrypted outputs differ. As a result, instead of having to guess 56 bits of a key (which requires a search through 2^{56} possible keys), would-be crackers could decipher the message after trying only 2^{46} keys or so—a 1000-fold improvement. More important, computers got faster. DES was being cracked by brute-force searches in which speedy computers simply tried every possible key. Last year, in response to a challenge by the San Jose–based cryptography company RSA Security, volunteers yoked nearly 100,000 PCs together via the Internet to decipher a DES-encrypted message. They succeeded in less than a day. To beef up security, wary DES users started running the algorithm three times with three different keys. NIST, however, decided that patches were not enough. In 1997, the institute called for a new standard; the AES contest was born.

Cryptographers from all over the world submitted candidate algorithms, from which NIST selected 15. Two years and a lot of code-cracking and skirmishing later, NIST narrowed the field to five finalists—and the international cryptographic community turned its attention to testing, and breaking, them.

On 13 and 14 April, participants in the

cryptanalytic demolition derby gathered in New York to present their results at the third AES conference, the final gathering before NIST chooses a standard late in the summer. "It's the last AES conference. I'm thankful," sighed NIST's Jim Foti. An overhead projector aimed at a screen at the front of the room greeted participants with an apt image: King Kong perched atop the Empire State Building, under attack by a swarm of biplanes.

As cryptographic algorithms go, the five rival contestants are 500-pound gorillas themselves. Each employs a key 128, 192, or 256 bits long—potentially much more secure than the 56-bit key of DES. In broad terms, they all do the same thing, taking a 128-bit chunk of text and jumbling it up and changing the bits so that they become unreadable. Each scrambling algorithm is publicly available; even if an eavesdropper knows exactly how the encrypting machine works, the knowledge is useless without the key.

An important feature of a secure cipher is that a tiny change in the cryptographic key makes a huge difference in the scrambled output. As a result, guessing part of a key doesn't give any information about the text or about the rest of the key. Ideally, even if an eavesdropper guesses 127 of the 128 bits in the key, the message decrypted with this "key" should be just as unreadable as if he used a random key. This small-key-changes-yield-huge-output-changes feature is called "nonlinearity."

Four of the five algorithms create nonlinearity the way DES does, through devices called "S-boxes." An S-box takes a string of ones and zeros and returns a different set; it's essentially a look-up table that converts one string to an unrelated one, turning small changes in input into large changes in output. IBM's entry, MARS, boasts an S-box–based cryptographic "core" surrounded by a "wrapper" of lighter weight scrambling subroutines that protect the core from direct cryptanalytic assault. Serpent, designed by Biham and partners in Britain and Norway, passes information through eight S-boxes over and over, cycling the text through many more "rounds" of manipulation than its competitors do before it spits out the encrypted data. Twofish, designed by Bruce Schneier of Counterpane and colleagues, changes the contents of the S-boxes depending on the cryptographic key, unlike the others, which have fixed S-boxes. Rijndael, the entry of two Belgian cryptographers, relies upon elegant mathematical ma-

nipulations of the data, arranged into a square; a single S-box adds nonlinearity.

The S-box–free algorithm is RC6, designed by RSA Security's Ron Rivest and other cryptographers in the United States and Britain. It takes slices of data and "rotates" them by cutting a chunk off one end and pasting it back on the other. The amount of rotation depends upon the data being rotated. Changing a single bit in a chunk of data tends to cause a change in rotation, altering the data quite a bit and changing how the data get rotated even more; small differences in data propagate and proliferate as the process repeats over and over, ensuring nonlinearity.

Choosing a winner among the finalists is no easy task. Even figuring out which algorithm runs fastest is almost an intractable problem, because of the huge number of different types of software and hardware the algorithm will run on. Rijndael appears to be the fastest overall, but most designers had to be content with a mixed showing, slow on some platforms and fast on others. Serpent, for example, is the quickest on field-programmable gate arrays—reconfigurable hardware devices—but the slowest on a Pentium. "Looking at all the performance requirements, we generally don't suck," says Twofish designer Schneier.

More important than speed is security. None of the algorithms has been broken, but some have been bloodied. Rijndael performs its mixing and jumbling operation 14 times before spitting out an answer. Cryptographers have figured out mathematical

tricks to crack an eight-rounded variant with a lot less effort than guessing all the possible keys. Nine of 16 rounds in MARS have been cracked, as have 15 of 20 in RC6. The attacks are still theoretical—no computer today could use them to crack a cipher in any reasonable amount of time—but they might be cause for worry in the future. "Attacks are improved all the time," Biham says. "If an algorithm has a very small security margin, it will be attacked." Biham's entry, Serpent, edges out Twofish for the distinction of most secure algorithm, although the difference is probably academic. "It's a bank vault versus a bank vault with a bit of kryptonite in case Superman walks by," jokes Nicholas Weaver, a cryptographer at the University of California, Berkeley.

Some conference participants, meanwhile, worried about a different type of assault: patent attacks. Intellectual property was not an issue when cryptography was mostly a government-only preserve, but now that commercial companies are in the cryptography game, the playing field has changed. Days before the conference opened, the software division of Hitachi Ltd. wrote to NIST claiming that it holds a U.S. patent on techniques used in four of the finalist algorithms. Whether or not the claim holds up, NIST is entering a "quagmire" of potential intellectual property disputes, says Josh Benaloh of Microsoft Research. The more popular and widely distributed a NIST-sponsored standard becomes, the messier and more expensive the legal battles might be. "I think it's far more likely than the cryptographic attack," Benaloh says—and potentially much harder for cryptographers to handle.

"We are not lawyers up here, you know," Schneier says. "I'm at a complete loss at how to deal with various patent laws." Others share his bewilderment. "This is a very real attack. This is a very significant attack," says James Hughes of StorageTek in Minneapolis. Even a hint of patent trouble should disqualify a contender, Hughes says: "If it happens, I suggest that NIST withdraw its suggestion immediately and pick another." For NIST, the possibility of patent troubles just serves to make a tough call even tougher. Any choice it makes will satisfy some parties and anger others. Shamir alone claims to have found an easy solution. "No one algorithm is head and shoulders above the rest," he says. "I suggest having a fair coin flip."   —CHARLES SEIFE

Code wars. Companies are vying to turn data into gibberish, but is the gibberish secure?

ILLUSTRATION: TERRY SMITH

TECHNOLOGY

HOT TRENDS & TECHNOLOGIES IN BRIEF

Authentication

BY RUSSELL KAY

WHO are you? Do you belong here? What rights do you have? And how do I know you're who you say you are?

Those are the essential questions that any effective security system must answer before a user can access a computer system, network or other protected resource. We think this is what a password system does, but passwords are only one part of an effective security system. That security system requires three separate elements — identification, authentication and authorization — that together make up what's called access control.

When you log into a computer or network, the first thing you're asked for is a user name or account name. But a user name offers little protection to the system. Therefore, the system also usually prompts you for a password, a form of authentication.

Authentication

The question, "How do I know you're who you say you are?," is in many ways, the most important one. Unless it's answered satisfactorily, identification is incomplete and no authorization can or should take place. But how does a system verify that a user is who he says he is? Simply entering your password doesn't prove it's you. Someone else could know your password.

The answer lies in a strong authentication process. Basically, the following three factors can be used to authenticate an individual:

1. Something the user knows. This is a reusable password, passphrase, personal identification number or a fact likely to be known only to the user, such as his mother's maiden name.

2. Something the user has. This could be a key, a magnetic-stripe card, a smart card or a specialized authentication device (called a token) that generates a one-time password or a specific response to a

---

DEFINITION

Authentication is the process through which the identity of a computer or network user is verified; it's the system that ensures that an individual is, in fact, who he claims to be. It's distinct from **identification** — determining whether an individual is known to the system — and from **authorization** — granting the user access to specific system resources based on his identity.

---

challenge presented by the server.

3. Something the user is. This depends on some inherent physical trait or characteristic. Often called biometrics, examples of this form of authentication include: fingerprints, retinal (eye) patterns, hand geometry, voice recognition, facial recognition, typing pattern recognition and signature dynamics (speed and pressure, not just the outline).

For more on biometrics, see "Give Your Computer the Finger" on page 78.

These authentication factors are listed here from weakest to strongest as determined by how difficult they are to forge or fake. By themselves, each of these methods offers some security. However, each has its own problems or weaknesses.

Anyone can enter a password and, historically, reusable passwords have been vulnerable to guessing, brute force and dictionary-based attacks.

The second means of authentication — something the user has — requires the user to possess an often difficult-to-replicate device. However this stronger protection also costs more (typically tens of dollars per device), and it requires contingency procedures in case a device is left at home, lost or stolen.

The third type of authentication — something the user is — is the most difficult to defeat, but it has other problems. Biometric identification methods are subject to two types of errors: false positives and false negatives. The first erroneously authenticates an individual who shouldn't be authenticated; the second denies an individual who should be authenticated. Neither error is desirable, and it's important to know and verify error rates when considering such a system.

Another problem is that permanent physical changes or temporary ailments or accidents can alter or render unreadable the measured characteristic. If you cut part of your fingertip, you've changed what the fingerprint reader sees. Put on a Band-Aid, and the reader can't see the fingerprint at all.

Finally, if the method is compromised, there's no way to give an individual a new identifying characteristic. You can issue a new password or security token, but you can't change his fingerprints or eye pattern.

Two-Factor Authentication

For greatly increased security, the approach preferred by experts is to use two of the three methods in combination — a process called two-factor authentication. For example, to use a security token that generates a one-time password, you may need to enter a personal identification number into the token itself. Similarly, a card-key can be used in combination with a biometric system.

This is essentially what happens when you check in at an airport ticket counter. You hand over your ticket, which identifies you. Then you show a photo ID of some kind. This is something you have with you, and it's biometric (something you are) in that the clerk has to determine that the photo on the card matches you.

Once a user has been identified and authenticated, what remains is to grant him access to whatever specific system resources have been approved. This authorization is usually accomplished by looking up that user's entry in an access control list that delineates specific rights and permissions. These can be based, among other things, on an individual's identity or job function, membership in a workgroup or other classification or time of day or day of week. ▶

Authentication via Security Token

SecurID

A hardware authentication device, or security token, provides greatly increased protection against spoofing or brute-force attacks. The time-synchronized SecurID card from RSA Security Inc. in Bedford, Mass., has an LCD screen that shows a string of numbers that changes every minute. The user types in his user name at log in, then the number shown on the card. The host system knows what that number is supposed to be for that user at that particular time. Some tokens don't show a number continuously but require the user to enter a PIN on the card itself before the number is displayed, thus providing two-factor authentication.

Challenge-Response Systems

With a token-based Challenge-Response system, the system displays a number (the challenge) when you log in. The user types this number into his token, which encrypts that to produce a second number (the response). The user enters the response into the computer. The host performs the same operation on the challenge, then compares its result to the user's response. If they match, the user is authenticated.
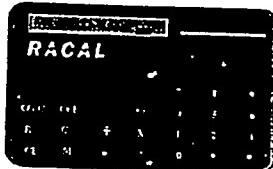
Give Your Computer the Finger

Do you secretly suspect that after they made you they broke the mold? According to the nine biometric security products we tested, you're right. By Howard Millman

WHEN YOU need better protection than a password system can give you, it may be time to consider using a biometric authentication device. In our tests, these devices proved affordable, reliable, easy to use and light-years ahead of passwords in boosting desktop, laptop and network access protection.

Not only are passwords easily compromised, they don't authenticate people — they merely authenticate passwords. Conversely, each fingerprint is unique. With biometrics, you'll never have to remember multiple, sometimes counterintuitive alphanumeric sequences. All you need to remember is to bring your finger or face, or how to sign your name.

Biometric devices measure one or more physical attributes. The most commonly used attribute is your fingerprint, but it can also be the shape of your face, the pattern of your eye's iris, your signature or the sound of your voice.

Devices exist to meet any degree of security and paranoia. For example, if you want ultrasecure access to the ICBM missile silo or an anthrax lab,

retinal-scan devices that read the pattern of blood vessels inside eyes are available. We confined our tests to noninvasive devices suited for use with computers and networks in a normal business environment.

"Biometrics have been around a long time while the vendors tried to get the technology and price right. Finally, fingerprint scanners are a here-and-now technology," says Chris Christensen, a security analyst at International Data Corp. in Framingham, Mass. Starting this summer, manufacturers like Compaq Computer Corp. will ship laptops equipped with biometric devices.

With prices dropping and accuracy increasing, the future looks promising for vendors. According to New York-based consulting firm International Biometric Group LLC, the market for biometric devices totaled $260 million last year. The company predicts a 30% to 40% annual growth rate.

All the products we tested are ready for use and were designed for existing machines. The devices cost between $60 and $395 and offer vastly increased security. All products were installed effortlessly, required no maintenance and delivered consistent accuracy.

Digital Persona Inc.'s U.are.U

Pro offers a major advantage over the other products — a single cable connection to the Universal Serial Bus (USB) port in machines running Windows 95/98, NT and 2000. The other devices require a connection to the parallel port, a power source (usually the keyboard connector cable) and a printer pass-through when a printer is connected to the same machine.

Aside from the convenience of a USB connector, hardware from the other vendors performed equally well. When used on a Windows NT network and integrated into NT's Security Access Manager, all the devices provided security far superior to a mere password.

Biometric vendors tend to sell either just the software engine, such as Identix Inc.'s Bio-Logon and Cyber-SIGN Inc.'s Cyber-SIGN, or the hardware, such as products from SCM Microsystems Inc. and Interlink Electronics Inc. Others vendors, such as Keyware Technologies and Digital Persona, offer both.

Other than Digital's Persona's custom-developed USB driver for Windows NT, we found no major differences in the ease of use, reliability or feature set in any of the software. All the vendors mentioned plan to release USB versions of their products for Windows 2000.

For information technology use, the device chosen should be based as much on price as on desktop space. Most fingerprint scanners and signature readers are stand-alone devices, but Key Tronic Corp.'s Key Tronic Secure Keyboard integrates a scanner into a keyboard. The

bundling saves space, and help desks may prefer built-in devices to those added on.

Since pressing a finger is slightly easier than writing your signature or mugging it up for a camera, fingerprint scanners have a slight edge in usability. All devices also allow password entry in the event that the biometric recognition fails, perhaps as the result of an accident or illness. To help prevent erroneous access denials, authorized users should register multiple fingers. ▶

Millman operates Data System Services LLC, a consultancy in Croton, N.Y. Contact him at hmillman@ibm.net.

U.are.U Pro

Hardware and software; fingerprint
Digital Persona Inc.
Redwood City, Calif.
www.digitalpersona.com
(650) 261-6070
$199 with client software
Server software: $29 per user

The all-around winner. Highly accurate with inexpensive, single-cable USB connectivity. The software shows its heritage as a onetime consumer product – it's entertaining and simple to understand. Users enroll and set up security via an overlay to Windows NT's Management Console. Utilities include a one-touch Internet log-in, which eliminates the need for passwords, and private space, a virtual drive that stores encrypted data.

BioTouch

Hardware; fingerprint
SCM Microsystems Inc.
Los Gatos, Calif.
www.scmmicro.com
(408) 370-4888
$179

BioLogon 2.02

Software; fingerprint
Identix Inc.
Sunnyvale, Calif.

IBM Operating System/2 Version 1.2

Profile Validation Hooks

November 22, 1989

This document is intended to provide additional external interface information that may be helpful to you in adapting or using OS/2. Some or all of the interfaces described in this document are not included in the formal definition of external interfaces as described in the *OS/2 Programming Tools and Information* library. The interfaces described in this document are specific to a particular version of OS/2 and are subject to change or elimination in future versions, at IBM's sole discretion, without notice to you. IBM does not represent or guarantee that compatibility of software or hardware which uses these interfaces can or will be maintained with future versions of OS/2.

This document could include technical inaccuracies or typographical errors. It is possible that this document may contain reference to, or information about, IBM products, programming or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming or services in your country.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Armonk NY 10504.

About This Document

This document gives a detailed technical description of the hooks that an application can use to hook calls that read from, write to, or query the profile files (also called the *initialization* files) in IBM Operating System/2 Standard Edition Version 1.2. Use this information in conjunction with that supplied in the *IBM OS/2 Version 1.2 Programming Tools and Information* library (part number 6024929).

The hooks are described in a *metalanguage*, in the same format as that in the *Presentation Manager Programming Reference*. Language bindings are given for IBM C/2 and IBM Macro Assembler/2. The hooks are not available in COBOL and FORTRAN.

For a description of the other data types, structures, and calls that are referred to in this document, see the *Presentation Manager Programming Reference*. For language-specific descriptions of these data types, structures, and calls, see the relevant Presentation Manager bindings reference in the *IBM OS/2 Version 1.2 Programming Tools and Information* library.

iv   Profile Validation Hooks

Contents

Introduction

The Presentation Manager of OS/2 Version 1.2 has two profile files. These are referred to as the *user profile* and the *system profile*. Profile files are also called *initialization* files.

The user profile is usually named OS2.INI, but the name is taken from the PROTSHELL statement in the CONFIG.SYS file. You can use the PrfReset call to change the file that is used as the user profile. The user profile contains the information that is specific to an individual user. Examples are:

- User preferences, such as screen colors and window-border width
- The list of programs held by the Desktop Manager
- Application settings.

The system profile is usually named OS2SYS.INI, but again the name is taken from the PROTSHELL statement in the CONFIG.SYS file. You *cannot* use the PrfReset call to change the file that is used as the system profile. The system profile contains the system and hardware information. Examples are:

- Names of printers and printer drivers
- Names of ports
- Names of printer queues
- The spooler path.

The system profile is new for OS/2 Version 1.2. By keeping system-specific information separate from the user-specific information, the networked use of OS/2 can be made transparent to the user.

The contents of the user and system profiles are defined in Appendix E of the *OS/2 Version 1.2 Presentation Manager Programming Reference: Volume 2.*

The Profile Validation Hooks

The profile validation hooks allow applications and system components to intercept (or *hook*) calls that read from, write to, or query the profile files. Hooks are described in the *OS/2 Version 1.2 Programming Guide.*

When an application makes one of these profile calls, the system component that handles the profile application programming interface (API) creates an *API packet*. This is a data structure that identifies the call and contains the parameters passed by the application. The API packet is then passed to the first profile hook chain: the Program List Entry hook. This chain allows an application or a system component to validate data that is being written to the profile. If an application or a system component depends on its data being correct, it can use this hook to reject invalid data. The final entry in this hook chain is part of OS/2 and it will update the profile or program list.

After the API packet has been processed (and the output parameters and return values have been updated), it is passed to the Program List Exit hook chain. This allows an application or a system component to examine the results of a

profile call. When this hook chain completes, control returns to the application that made the original call.

The WinSetHook and WinReleaseHook calls now accept two new hook types:

- HK_PLIST_ENTRY
- HK_PLIST_EXIT.

These hook types are defined in the include file called PMWIN.H.

Restrictions on Use

The following restrictions apply:

- Routines called from either of the program list hook chains cannot make program-list calls or profile calls.

- Hook procedures are called at, and should execute at, ring 3. They should not attempt transitions to ring 2. This is because the profile API is callable from both ring 2 and ring 3. If the profile API is called at ring 2, it executes a DosCallBack to ensure that the hook procedures are called from ring 3. Therefore a hook procedure may be called within a DosCallBack environment.

- The program-list API cannot be hooked during the system-install process.

- In normal operation, the program-list hooks are set up as part of system initialization. Therefore any profile calls that are made before system initialization is complete may not be hooked.

ProgramListEntryHook –
Program List Entry Hook

This hook chain is called every time a program-list call or initialization-file call is invoked by an application. It is called before the call is executed.

ProgramListEntryHook (hab, HookParms, Suppress, *Processed*)

Parameters

hab *(HAB)* – input
Anchor-block handle.

HookParms *(PRFHOOKPARMS)* – input
Profile hook parameters.

These identify the call and give its parameters and return code. The *PRFHOOKPARMS* data type is described on page 7.

Suppress *(BOOL)* – output
Suppress indicator.

TRUE   If this is set by any hook procedure, no further processing of the call is done.

FALSE   If all hook procedures set Suppress to FALSE, the call is processed normally.

Processed *(BOOL)* – return
Processed indicator:

TRUE   The next hook in the chain is not called.
FALSE   The next hook in the chain is called.

Remarks

This hook, together with its counterpart ProgramListExitHook (described on page 6), allows applications or system components to:

1. Re-implement the initialization file and program list completely. This is not recommended.

2. Implement the initialization file and program list partially, whilst retaining the existing implementation. For example, read-only requests could be satisfied from memory, rather than from disk.

3. Redirect initialization-file operations on a particular group to an alternative (opened) profile. For example, in a multiple-user environment, a LAN program might choose to redirect profile groups that are hardware-dependent, rather than user-dependent, to the system-initialization file.

C/2 Binding

```
BOOL            fProcessed = ProgramListEntryHook
                                 (hab, pprfhkpHookParms, pfSuppress)

HAB             hab;                /* Anchor-block handle    */
PPRFHOOKPARMS   pprfhkpHookParms;   /* Profile hook parameters */
PBOOL           pfSuppress;         /* Suppress indicator     */

BOOL            fProcessed;         /* Processed indicator    */
```

Macro Assembler/2 Binding

```
EXTRN ProgramListEntryHook:FAR

PUSH    DWORD   hab                 ;Anchor-block handle
PUSH@   OTHER   prfhkpHookParms     ;Profile hook parameters
PUSH@   WORD    fSuppress           ;Suppress indicator
CALL ProgramListEntryHook

Returns WORD    fProcessed          ;Processed indicator
```

5

ProgramListExitHook –
Program List Exit Hook

This hook chain is called every time a program-list call or initialization-file call is invoked by an application. It is called after the call is executed.

ProgramListExitHook (hab, HookParms, *Processed*)

Parameters

hab *(HAB)* – input
Anchor-block handle.

HookParms *(PRFHOOKPARMS)* – input
Profile hook parameters.

These identify the call and give its parameters and return code. The *PRFHOOKPARMS* data type is described on page 7.

Processed *(BOOL)* – return
Processed indicator:

TRUE   The next hook in the chain is not called.
FALSE   The next hook in the chain is called.

Remarks

See ProgramListEntryHook on page 4.

C/2 Binding

```
BOOL           fProcessed = ProgramListExitHook
                                   (hab, pprfhkpHookParms)

HAB            hab;              /* Anchor-block handle    */
PPRFHOOKPARMS  pprfhkpHookParms; /* Profile hook parameters */

BOOL           fProcessed;    ·  : /* Processed indicator     */
```

Macro Assembler/2 Binding

```
EXTRN ProgramListExitHook:FAR

PUSH   DWORD  hab                ;Anchor-block handle
PUSH@  OTHER  prfhkpHookParms    ;Profile hook parameters
CALL ProgramListExitHook

Returns WORD fProcessed          ;Processed indicator
```

Metalanguage Data Types

PRFHOOKPARMS   Profile hook parameters structure.

This structure is used for the HK_PLIST_ENTRY hook (see ProgramListEntryHook on page 4), and the HK_PLIST_EXIT hook (see ProgramListExitHook on page 6).

numbytes *(COUNT4B)*
Length of API packet structure.

api *(ULONG)*
API call identifier.

PLSTAPI_PRFQUERYPROGRAMTITLES	PrfQueryProgramTitles, WinQueryProgramTitles.
PLSTAPI_PRFADDPROGRAM	PrfAddProgram, WinAddProgram.
PLSTAPI_PRFCHANGEPROGRAM	PrfChangeProgram.
PLSTAPI_PRFQUERYDEFINITION	PrfQueryDefinition, WinQueryDefinition.
PLSTAPI_PRFREMOVEPROGRAM	PrfRemoveProgram.
PLSTAPI_PRFQUERYPROGRAMHANDLE	PrfQueryProgramHandle.
PLSTAPI_PRFCREATEGROUP	PrfCreateGroup, WinCreateGroup.
PLSTAPI_PRFDESTROYGROUP	PrfDestroyGroup.
PLSTAPI_PRFQUERYPROGRAMTYPE	PrfQueryProgramCategory.
INIAPI_PRFQUERYPROFILESTRING	PrfQueryProfileString, WinQueryProfileString.
INIAPI_PRFWRITEPROFILESTRING	PrfWriteProfileString, WinWriteProfileString.
INIAPI_PRFQUERYPROFILESIZE	PrfQueryProfileSize, WinQueryProfileSize.
INIAPI_PRFQUERYPROFILEDATA	PrfQueryProfileData, WinQueryProfileData.
INIAPI_PRFQUERYPROFILEINT	PrfQueryProfileInt, WinQueryProfileInt.
INIAPI_PRFWRITEPROFILEDATA	PrfWriteProfileData, WinWriteProfileData.
INIAPI_PRFOPENPROFILE	PrfOpenProfile.
INIAPI_PRFCLOSEPROFILE	PrfCloseProfile.
INIAPI_PRFRESET	PrfReset.
INIAPI_PRFQUERYPROFILE	PrfQueryProfile.

apiparms *(STORAGE)*
API parameters packet. The descriptions of these data types follow that of *PRFHOOKPARMS*.

PRFQUERYPROGRAMTITLESPARMS	PrfQueryProgramTitles, WinQueryProgramTitles.
PRFADDPROGRAMPARMS	PrfAddProgram, WinAddProgram.
PRFCHANGEPROGRAMPARMS	PrfChangeProgram.
PRFQUERYDEFINITIONPARMS	PrfQueryDefinition, WinQueryDefinition.
PRFREMOVEPROGRAMPARMS	PrfRemoveProgram.
PRFQUERYPROGRAMHANDLEPARMS	PrfQueryProgramHandle.

PRFCREATEGROUPPARMS	PrfCreateGroup, WinCreateGroup.
PRFDESTROYGROUPPARMS	PrfDestroyGroup.
PRFQUERYPROGRAMCATEGORYPARMS	PrfQueryProgramCategory.
PRFQUERYPROFILESTRINGPARMS	PrfQueryProfileString, WinQueryProfileString.
PRFWRITEPROFILESTRINGPARMS	PrfWriteProfileString, WinWriteProfileString.
PRFQUERYPROFILESIZEPARMS	PrfQueryProfileSize, WinQueryProfileSize.
PRFQUERYPROFILEDATAPARMS	PrfQueryProfileData, WinQueryProfileData.
PRFQUERYPROFILEINTPARMS	PrfQueryProfileInt, WinQueryProfileInt.
PRFWRITEPROFILEDATAPARMS	PrfWriteProfileData, WinWriteProfileData.
PRFOPENPROFILEPARMS	PrfOpenProfile.
PRFCLOSEPROFILEPARMS	PrfCloseProfile.
PRFRESETPARMS	PrfReset.
PRFQUERYPROFILEPARMS	PrfQueryProfile.

PRFADDPROGRAMPARMS

API packet containing the parameters of a PrfAddProgram call.

hini *(HINI)*
Initialization-file handle.

details *(PROGDETAILS)*
Program details.

group *(HPROGRAM)*
Handle of group.

program *(HPROGRAM)*
Program handle.

errorid *(ERRORID)*
Error identity.

PRFCHANGEPROGRAMPARMS

API packet containing the parameters of a PrfChangeProgram call.

hini *(HINI)*
Initialization-file handle.

program *(HPROGRAM)*
Program handle.

details *(PROGDETAILS)*
Program details.

success *(BOOL)*
Success indicator.

errorid *(ERRORID)*
Error identity.

PRFCLOSEPROFILEPARMS

API packet containing the parameters of a PrfCloseProfile call.

hini *(HINI)*
Initialization-file handle.

success *(BOOL)*
Success indicator.

errorid *(ERRORID)*
Error identity.

PRFCREATEGROUPPARMS

API packet containing the parameters of a PrfCreateGroup call.

hini *(HINI)*
Initialization-file handle.

title *(STRL)*
Title of the new group.

visibility *(BIT8)*
Visibility control.

group *(HPROGRAM)*
Program-group handle.

errorid *(ERRORID)*
Error identity.

PRFDESTROYGROUPPARMS

API packet containing the parameters of a PrfDestroyGroup call.

hini *(HINI)*
Initialization-file handle.

group *(HPROGRAM)*
Group handle.

success *(BOOL)*
Success indicator.

errorid *(ERRORID)*
Error identity.

PRFOPENPROFILEPARMS

API packet containing the parameters of a PrfOpenProfile call.

hab *(HAB)*
Anchor-block handle.

filename *(STRL)*
Profile file name.

hini *(HINI)*
Initialization-file handle.

errorid *(ERRORID)*
Error identity.

PRFQUERYDEFINITIONPARMS

API packet containing the parameters of a PrfQueryDefinition call.

hini *(HINI)*
Initialization-file handle.

program *(HPROGRAM)*
Handle of program.

details *(PROGDETAILS)*
Program details.

maxlen *(LENGTH4)*
Buffer length.

retlen *(LENGTH4)*
Length of returned data.

errorid *(ERRORID)*
Error identity.

PRFQUERYPROFILEDATAPARMS

API packet containing the parameters of a PrfQueryProfileData call.

hini *(HINI)*
Initialization-file handle.

app *(STRL)*
Application name.

key *(STRL)*
Key name.

value *(STORAGE)*
Value data.

size *(LENGTH4)*
Size of value data.

success *(BOOL)*
Success indicator.

errorid *(ERRORID)*
Error identity.

PRFQUERYPROFILEINTPARMS

API packet containing the parameters of a PrfQueryProfileInt call.

hini *(HINI)*
Initialization-file handle.

app *(STRL)*
Application name.

key *(STRL)*
Key name.

default *(SHORT)*
Key name.

result *(SHORT)*
Result.

errorid *(ERRORID)*
Error identity.

PRFQUERYPROFILEPARMS

API packet containing the parameters of a PrfQueryProfile call.

hab *(HAB)*
Anchor-block handle.

profile *(PRFPROFILE)*
Profile-names structure.

success *(BOOL)*   .
Success indicator.

errorid *(ERRORID)*
Error identity.

PRFQUERYPROFILESIZEPARMS

API packet containing the parameters of a PrfQueryProfileSize call.

hini *(HINI)*
Initialization-file handle.

app *(STRL)*
Application name.

key *(STRL)*
Key name.

datalen *(LENGTH4)*
Data length.

success *(BOOL)*
Success indicator.

errorid *(ERRORID)*
Error identity.

PRFQUERYPROFILESTRINGPARMS

API packet containing the parameters of a PrfQueryProfileString call.

hini *(HINI)*
Initialization-file handle.

app *(STRL)*
Application name.

key *(STRL)*
Key name.

default *(STRL)*
Default string.

profilestring *(STORAGE)*
Profile string.

maxlen *(LENGTH4)*
Maximum string length.

retlen *(LENGTH4)*
String length returned.

errorid *(ERRORID)*
Error identity.

PRFQUERYPROGRAMCATEGORYPARMS

API packet containing the parameters of a PrfQueryProgramCategory call.

hini *(HINI)*
Initialization-file handle.

exe *(STRL)*
Executable-file name.

category *(PROGCATEGORY)*
Program category.

errorid *(ERRORID)*
Error identity.

13

PRFQUERYPROGRAMHANDLEPARMS

> API packet containing the parameters of a PrfQueryProgramHandle call.

hini *(HINI)*
Initialization-file handle.

exe *(STRL)*
Executable-file name.

progarray *(HPROGARRAY)*
Array of program handles.

maxlen *(LENGTH4)*
Maximum length of array (in bytes).

count *(COUNT4)*
Number of program handles returned.

retlen *(LENGTH4)*
Length of returned data.

errorid *(ERRORID)*
Error identity.

PRFQUERYPROGRAMTITLESPARMS

> API packet containing the parameters of a PrfQueryProgramTitles call.

hini *(HINI)*
Initialization-file handle.

group *(HPROGRAM)*
Handle of program or group.

titles *(PROGTITLE)*
Program information buffer.

maxlen *(LENGTH4)*
Buffer length.

count *(COUNT4)*
Number of structures returned.

retlen *(LENGTH4)*
Length of returned data.

errorid *(ERRORID)*
Error identity.

PRFREMOVEPROGRAMPARMS
API packet containing the parameters of a PrfRemoveProgram call.

hini *(HINI)*
Initialization-file handle.

program *(HPROGRAM)*
Program handle.

success *(BOOL)*
Success indicator.

errorid *(ERRORID)*
Error identity.

PRFRESETPARMS API packet containing the parameters of a PrfReset call.

hab *(HAB)*
Anchor-block handle.

profile *(PRFPROFILE)*
Profile-names structure.

success *(BOOL)*
Success indicator.

errorid *(ERRORID)*
Error identity.

PRFWRITEPROFILEDATAPARMS
API packet containing the parameters of a PrfWriteProfileData call.

hini *(HINI)*
Initialization-file handle.

app *(STRL)*
Application name.

key *(STRL)*
Key name.

value *(STORAGE)*
Value data.

size *(LENGTH4)*
Size of value data.

success *(BOOL)*
Success indicator.

errorid *(ERRORID)*
Error identity.

API packet containing the parameters of a PrfWriteProfileString call.

hini *(HINI)*
Initialization-file handle.

app *(STRL)*
Application name.

key *(STRL)*
Key name.

value *(STRL)*
Text string.

success *(BOOL)*
Success indicator.

errorid *(ERRORID)*
Error identity.

C/2 Data Types

PRFHOOKPARMS Profile hook parameters structure.

This structure is used for the HK_PLIST_ENTRY hook (see ProgramListEntryHook on page 4), and the HK_PLIST_EXIT hook (see ProgramListExitHook on page 6).

```
typedef struct _PRFHOOKPARMS {
   ULONG      NumBytes;    /* Length of API packet structure */
   ULONG      Api;         /* API call identifier            */
   PVOID      pApiParms;   /* API parameters packet          */
} PRFHOOKPARMS;
```

PRFADDPROGRAMPARMS

API packet containing the parameters of a PrfAddProgram call.

```
typedef struct _PRFADDPROGRAMPARMS {
   HINI           hini;         /* Initialization-file handle */
   PPROGDETAILS   pDetails;     /* Program details            */
   HPROGRAM       hprogGroup;   /* Handle of group            */
   PHPROGRAM      phprog;       /* Program handle             */
   PERRORID       pErrorId;     /* Error identity             */
} PRFADDPROGRAMPARMS;
```

PRFCHANGEPROGRAMPARMS

API packet containing the parameters of a PrfChangeProgram call.

```
typedef struct _PRFCHANGEPROGRAMPARMS {
   HINI           hini;         /* Initialization-file handle */
   HPROGRAM       hprog;        /* Program handle             */
   PPROGDETAILS   pDetails;     /* Program details            */
   PBOOL          pfSuccess;    /* Success indicator          */
   PERRORID       pErrorId;     /* Error identity             */
} PRFCHANGEPROGRAMPARMS;
```

PRFCLOSEPROFILEPARMS

API packet containing the parameters of a PrfCloseProfile call.

```
typedef struct _PRFCLOSEPROFILEPARMS {
HINI          hini;          /* Initialization-file handle */
PBOOL         pfSuccess;     /* Success indicator          */
PERRORID      pErrorId;      /* Error identity             */
} PRFCLOSEPROFILEPARMS;
```

PRFCREATEGROUPPARMS

API packet containing the parameters of a PrfCreateGroup call.

```
typedef struct _PRFCREATEGROUPPARMS {
HINI          hini;          /* Initialization-file handle */
PSZ           pszTitle;      /* Title of the new group     */
UCHAR         chVisibility;  /* Visibility control         */
PHPROGRAM     phprogGroup;   /* Program-group handle       */
PERRORID      pErrorId;      /* Error identity             */
} PRFCREATEGROUPPARMS;
```

PRFDESTROYGROUPPARMS

API packet containing the parameters of a PrfDestroyGroup call.

```
typedef struct _PRFDESTROYGROUPPARMS {
HINI          hini;          /* Initialization-file handle */
HPROGRAM      hprogGroup;    /* Group handle               */
PBOOL         pfSuccess;     /* Success indicator          */
PERRORID      pErrorId;      /* Error identity             */
} PRFDESTROYGROUPPARMS;
```

PRFOPENPROFILEPARMS

API packet containing the parameters of a PrfOpenProfile call.

```
typedef struct _PRFOPENPROFILEPARMS {
HAB           hab;           /* Anchor-block handle        */
PSZ           pszFileName;   /* Profile file name          */
PHINI         phini;         /* Initialization-file handle */
PERRORID      pErrorId;      /* Error identity             */
} PRFOPENPROFILEPARMS;
```

PRFQUERYDEFINITIONPARMS

API packet containing the parameters of a PrfQueryDefinition call.

```
typedef struct _PRFQUERYDEFINITIONPARMS {
HINI          hini;          /* Initialization-file handle */
HPROGRAM      hprog;         /* Handle of program          */
PPROGDETAILS  pDetails;      /* Program details            */
ULONG         cchBufferMax;  /* Buffer length              */
PULONG        pulLength;     /* Length of returned data    */
PERRORID      pErrorId;      /* Error identity             */
} PRFQUERYDEFINITIONPARMS;
```

PRFQUERYPROFILEDATAPARMS

API packet containing the parameters of a PrfQueryProfileData call.

```
typedef struct _PRFQUERYPROFILEDATAPARMS {
    HINI        hini;           /* Initialization-file handle */
    PSZ         pszApp;         /* Application name           */
    PSZ         pszKey;         /* Key name                   */
    PVOID       pBuffer;        /* Value data                 */
    PULONG      pulBufferMax;   /* Size of value data         */
    PBOOL       pfSuccess;      /* Success indicator          */
    PERRORID    pErrorId;       /* Error identity             */
} PRFQUERYPROFILEDATAPARMS;
```

PRFQUERYPROFILEINTPARMS

API packet containing the parameters of a PrfQueryProfileInt call.

```
typedef struct _PRFQUERYPROFILEINTPARMS {
    HINI        hINI;           /* Initialization-file handle */
    PSZ         pApplName;      /* Application name           */
    PSZ         pKeyName;       /* Key name                   */
    SHORT       Default;        /* Key name                   */
    PSHORT      pResult;        /* Result                     */
    PERRORID    pErrorId;       /* Error identity             */
} PRFQUERYPROFILEINTPARMS;
```

PRFQUERYPROFILEPARMS

API packet containing the parameters of a PrfQueryProfile call.

```
typedef struct _PRFQUERYPROFILEPARMS {
    HAB          hab;           /* Anchor-block handle      */
    PPRFPROFILE  pProfile;      /* Profile-names structure  */
    PBOOL        pfSuccess;     /* Success indicator        */
    PERRORID     pErrorId;      /* Error identity           */
} PRFQUERYPROFILEPARMS;
```

PRFQUERYPROFILESIZEPARMS

API packet containing the parameters of a PrfQueryProfileSize call.

```
typedef struct _PRFQUERYPROFILESIZEPARMS {
    HINI        hini;           /* Initialization-file handle */
    PSZ         pszApp;         /* Application name           */
    PSZ         pszKey;         /* Key name                   */
    PULONG      pulDataLen;     /* Data length                */
    PBOOL       pfSuccess;      /* Success indicator          */
    PERRORID    pErrorId;       /* Error identity             */
} PRFQUERYPROFILESIZEPARMS;
```

PRFQUERYPROFILESTRINGPARMS

API packet containing the parameters of a PrfQueryProfileString call.

```
typedef struct _PRFQUERYPROFILESTRINGPARMS {
HINI            hini;          /* Initialization-file handle */
PSZ             pszApp;        /* Application name           */
PSZ             pszKey;        /* Key name                   */
PSZ             pszDefault;    /* Default string             */
PVOID           pBuffer;       /* Profile string             */
ULONG           cchBufferMax;  /* Maximum string length      */
PULONG          pulLength;     /* String length returned     */
PERRORID        pErrorId;      /* Error identity             */
} PRFQUERYPROFILESTRINGPARMS;
```

PRFQUERYPROGRAMCATEGORYPARMS

API packet containing the parameters of a PrfQueryProgramCategory call.

```
typedef struct _PRFQUERYPROGRAMCATEGORYPARMS {
HINI               hini;        /* Initialization-file handle */
PSZ                pszExe;      /* Executable-file name       */
PPROGCATEGORY      pCategory;   /* Program category           */
PERRORID           pErrorId;    /* Error identity             */
} PRFQUERYPROGRAMCATEGORYPARMS;
```

PRFQUERYPROGRAMHANDLEPARMS

API packet containing the parameters of a PrfQueryProgramHandle call.

```
typedef struct _PRFQUERYPROGRAMHANDLEPARMS {
HINI            hini;          /* Initialization-file handle */
PSZ             pszExe;        /* Executable-file name       */
PHPROGARRAY     phprogArray;   /* Array of program handles   */
ULONG           cchBufferMax;  /* Maximum length of array    */
                               /* (in bytes)                 */
PULONG          pulCount;      /* Number of program handles  */
                               /* returned                   */
PULONG          pulLength;     /* Length of returned data    */
PERRORID        pErrorId;      /* Error identity             */
} PRFQUERYPROGRAMHANDLEPARMS;
```

PRFQUERYPROGRAMTITLESPARMS

API packet containing the parameters of a PrfQueryProgramTitles call.

```
typedef struct _PRFQUERYPROGRAMTITLESPARMS {
HINI            hini;          /* Initialization-file handle  */
HPROGRAM        hprogGroup;    /* Handle of program or group  */
PPROGTITLE      pTitles;       /* Program information buffer  */
ULONG           cchBufferMax;  /* Buffer length               */
PULONG          pulCount;      /* Number of structures returned */
PULONG          pulLength;     /* Length of returned data     */
PERRORID        pErrorId;      /* Error identity              */
} PRFQUERYPROGRAMTITLESPARMS;
```

PRFREMOVEPROGRAMPARMS

API packet containing the parameters of a PrfRemoveProgram call.

```
typedef struct _PRFREMOVEPROGRAMPARMS {
HINI            hini;         /* Initialization-file handle */
HPROGRAM        hprog;        /* Program handle             */
PBOOL           pfSuccess;    /* Success indicator          */
PERRORID        pErrorId;     /* Error identity             */
} PRFREMOVEPROGRAMPARMS;
```

PRFRESETPARMS

API packet containing the parameters of a PrfReset call.

```
typedef struct _PRFRESETPARMS {
HAB             hab;          /* Anchor-block handle        */
PPRFPROFILE     pProfile;     /* Profile-names structure    */
PBOOL           pfSuccess;    /* Success indicator          */
PERRORID        pErrorId;     /* Error identity             */
} PRFRESETPARMS;
```

PRFWRITEPROFILEDATAPARMS

API packet containing the parameters of a PrfWriteProfileData call.

```
typedef struct _PRFWRITEPROFILEDATAPARMS {
HINI            hini;         /* Initialization-file handle */
PSZ             pszApp;       /* Application name           */
PSZ             pszKey;       /* Key name                   */
PVOID           pData;        /* Value data                 */
ULONG           cchDataLen;   /* Size of value data         */
PBOOL           pfSuccess;    /* Success indicator          */
PERRORID        pErrorId;     /* Error identity             */
} PRFWRITEPROFILEDATAPARMS;
```

PRFWRITEPROFILESTRINGPARMS

API packet containing the parameters of a PrfWriteProfileString call.

```
typedef struct _PRFWRITEPROFILESTRINGPARMS {
HINI            hini;         /* Initialization-file handle */
PSZ             pszApp;       /* Application name           */
PSZ             pszKey;       /* Key name                   */
PSZ             pszData;      /* Text string                */
PBOOL           pfSuccess;    /* Success indicator          */
PERRORID        pErrorId;     /* Error identity             */
} PRFWRITEPROFILESTRINGPARMS;
```

C/2 Header File

```
/* Prf api packet typedef */
typedef struct _PRFHOOKPARMS {
    ULONG NumBytes;
    ULONG Api;
    PVOID pApiParms;
} PRFHOOKPARMS;

typedef PRFHOOKPARMS FAR *PPRFHOOKPARMS;
/*********************************************************/
/*  INI file access API Packet identifiers   'Prf'       */
/*                                                       */
/*  Note these identifiers must not overlap the corresponding */
/*  set of INIAPI_WINXXXXXXXXX and PLSTAPI_WINXXXXXXXXXXXX    */
/*  values                                               */
/*********************************************************/

#define INIAPI_PRFQUERYPROFILEINT            0x0030
#define INIAPI_PRFQUERYPROFILESTRING         0x0031
#define INIAPI_PRFWRITEPROFILESTRING         0x0032
#define INIAPI_PRFQUERYPROFILESIZE           0x0033
#define INIAPI_PRFQUERYPROFILEDATA           0x0034
#define INIAPI_PRFWRITEPROFILEDATA           0x0035
#define INIAPI_PRFOPENPROFILE                0x0036
#define INIAPI_PRFCLOSEPROFILE               0x0037
#define INIAPI_PRFRESET                      0x0038
#define INIAPI_PRFQUERYPROFILE               0x0039


#ifdef INCL_WINPROGRAMLIST
/*********************************************************/
/*  Program List 'Prf' API packet identifiers            */
/*                                                       */
/*  Note these identifiers must not overlap the          */
/*  corresponding set of INIAPI_WINXXXXXXXXX             */
/*  and PLSTAPI_WINXXXXXXXXXXXX values                   */
/*********************************************************/
#define PLSTAPI_PRFQUERYPROGRAMTITLES        0x0022
#define PLSTAPI_PRFADDPROGRAM                0x0023
#define PLSTAPI_PRFCHANGEPROGRAM             0x0024
#define PLSTAPI_PRFQUERYDEFINITION           0x0025
#define PLSTAPI_PRFREMOVEPROGRAM             0x0026
#define PLSTAPI_PRFQUERYPROGRAMHANDLE        0x0027
#define PLSTAPI_PRFCREATEGROUP               0x0028
#define PLSTAPI_PRFDESTROYGROUP              0x0029
#define PLSTAPI_PRFQUERYPROGRAMCATEGORY      0x002A

#define HK_PLIST_ENTRY       9
    /* BOOL EXPENTRY ProgramListEntryHook
            (HAB hab,                        ** installer's hab      **
            PPRFHOOKPARMS pProfileHookParams, ** data                **
            PBOOL fNoExecute);               ** stop hook processing */

#define HK_PLIST_EXIT        10
    /* BOOL EXPENTRY ProgramListExitHook
            (HAB hab,                        ** installer's hab      **
            PPRFHOOKPARMS pProfileHookParams);** data               */
```

Macro Assembler/2 Data Types

PRFHOOKPARMS   Profile hook parameters structure.

This structure is used for the HK_PLIST_ENTRY hook (see ProgramListEntryHook on page 4), and the HK_PLIST_EXIT hook (see ProgramListExitHook on page 6).

```
PRFHOOKPARMS struc
    prfhkp_NumBytes        dd    ?    ;Length of API packet structure
    prfhkp_Api             dd    ?    ;API call identifier
    prfhkp_pApiParms       dd    ?    ;API parameters packet
PRFHOOKPARMS ends
```

PRFADDPROGRAMPARMS

API packet containing the parameters of a PrfAddProgram call.

```
PRFADDPROGRAMPARMS struc
    prfadp_hini            dd    ?    ;Initialization-file handle
    prfadp_pDetails        dd    ?    ;Program details
    prfadp_hprogGroup      dd    ?    ;Handle of group
    prfadp_phprog          dd    ?    ;Program handle
    prfadp_pErrorId        dd    ?    ;Error identity
PRFADDPROGRAMPARMS ends
```

PRFCHANGEPROGRAMPARMS

API packet containing the parameters of a PrfChangeProgram call.

```
PRFCHANGEPROGRAMPARMS struc
    prfchp_hini            dd    ?    ;Initialization-file handle
    prfchp_hprog           dd    ?    ;Program handle
    prfchp_pDetails        dd    ?    ;Program details
    prfchp_pfSuccess       dd    ?    ;Success indicator
    prfchp_pErrorId        dd    ?    ;Error identity
PRFCHANGEPROGRAMPARMS ends
```

PRFCLOSEPROFILEPARMS

API packet containing the parameters of a PrfCloseProfile call.

```
PRFCLOSEPROFILEPARMS struc
    prfcp_hini             dd    ?    ;Initialization-file handle
    prfcp_pfSuccess        dd    ?    ;Success indicator
    prfcp_pErrorId         dd    ?    ;Error identity
PRFCLOSEPROFILEPARMS ends
```

PRFCREATEGROUPPARMS

API packet containing the parameters of a PrfCreateGroup call.

```
PRFCREATEGROUPPARMS struc
    prfcg_hini             dd    ?    ;Initialization-file handle
    prfcg_pszTitle         dd    ?    ;Title of the new group
    prfcg_chVisibility     db    ?    ;Visibility control
    prfcg_phprogGroup      dd    ?    ;Program-group handle
    prfcg_pErrorId         dd    ?    ;Error identity
PRFCREATEGROUPPARMS ends
```

PRFDESTROYGROUPPARMS

API packet containing the parameters of a PrfDestroyGroup call.

```
PRFDESTROYGROUPPARMS struc
    prfdg_hini          dd   ?   ;Initialization-file handle
    prfdg_hprogGroup    dd   ?   ;Group handle
    prfdg_pfSuccess     dd   ?   ;Success indicator
    prfdg_pErrorId      dd   ?   ;Error identity
PRFDESTROYGROUPPARMS ends
```

PRFOPENPROFILEPARMS

API packet containing the parameters of a PrfOpenProfile call.

```
PRFOPENPROFILEPARMS struc
    prfop_hab           dd   ?   ;Anchor-block handle
    prfop_pszFileName   dd   ?   ;Profile file name
    prfop_phini         dd   ?   ;Initialization-file handle
    prfop_pErrorId      dd   ?   ;Error identity
PRFOPENPROFILEPARMS ends
```

PRFQUERYDEFINITIONPARMS

API packet containing the parameters of a PrfQueryDefinition call.

```
PRFQUERYDEFINITIONPARMS struc
    prfqd_hini          dd   ?   ;Initialization-file handle
    prfqd_hprog         dd   ?   ;Handle of program
    prfqd_pDetails      dd   ?   ;Program details
    prfqd_cchBufferMax  dd   ?   ;Buffer length
    prfqd_pulLength     dd   ?   ;Length of returned data
    prfqd_pErrorId      dd   ?   ;Error identity
PRFQUERYDEFINITIONPARMS ends
```

PRFQUERYPROFILEDATAPARMS

API packet containing the parameters of a PrfQueryProfileData call.

```
PRFQUERYPROFILEDATAPARMS struc
    prfqpd_hini         dd   ?   ;Initialization-file handle
    prfqpd_pszApp       dd   ?   ;Application name
    prfqpd_pszKey       dd   ?   ;Key name
    prfqpd_pBuffer      dd   ?   ;Value data
    prfqpd_pulBufferMax dd   ?   ;Size of value data
    prfqpd_pfSuccess    dd   ?   ;Success indicator
    prfqpd_pErrorId     dd   ?   ;Error identity
PRFQUERYPROFILEDATAPARMS ends
```

PRFQUERYPROFILEINTPARMS

API packet containing the parameters of a PrfQueryProfileInt call.

```
PRFQUERYPROFILEINTPARMS struc
    prfqpi_hINI         dd   ?   ;Initialization-file handle
    prfqpi_pApplName    dd   ?   ;Application name
    prfqpi_pKeyName     dd   ?   ;Key name
    prfqpi_Default      dw   ?   ;Key name
    prfqpi_pResult      dd   ?   ;Result
    prfqpi_pErrorId     dd   ?   ;Error identity
PRFQUERYPROFILEINTPARMS ends
```

PRFQUERYPROFILEPARMS

API packet containing the parameters of a PrfQueryProfile call.

```
PRFQUERYPROFILEPARMS struc
    prfqp_hab           dd  ?   ;Anchor-block handle
    prfqp_pProfile      dd  ?   ;Profile-names structure
    prfqp_pfSuccess     dd  ?   ;Success indicator
    prfqp_pErrorId      dd  ?   ;Error identity
PRFQUERYPROFILEPARMS ends
```

PRFQUERYPROFILESIZEPARMS

API packet containing the parameters of a PrfQueryProfileSize call.

```
PRFQUERYPROFILESIZEPARMS struc
    prfqpsz_hini        dd  ?   ;Initialization-file handle
    prfqpsz_pszApp      dd  ?   ;Application name
    prfqpsz_pszKey      dd  ?   ;Key name
    prfqpsz_pulDataLen  dd  ?   ;Data length
    prfqpsz_pfSuccess   dd  ?   ;Success indicator
    prfqpsz_pErrorId    dd  ?   ;Error identity
PRFQUERYPROFILESIZEPARMS ends
```

PRFQUERYPROFILESTRINGPARMS

API packet containing the parameters of a PrfQueryProfileString call.

```
PRFQUERYPROFILESTRINGPARMS struc
    prfqpst_hini        dd  ?   ;Initialization-file handle
    prfqpst_pszApp      dd  ?   ;Application name
    prfqpst_pszKey      dd  ?   ;Key name
    prfqpst_pszDefault  dd  ?   ;Default string
    prfqpst_pBuffer     dd  ?   ;Profile string
    prfqpst_cchBufferMax dd ?   ;Maximum string length
    prfqpst_pulLength   dd  ?   ;String length returned
    prfqpst_pErrorId    dd  ?   ;Error identity
PRFQUERYPROFILESTRINGPARMS ends
```

PRFQUERYPROGRAMCATEGORYPARMS

API packet containing the parameters of a PrfQueryProgramCategory call.

```
PRFQUERYPROGRAMCATEGORYPARMS struc
    prfqpc_hini         dd  ?   ;Initialization-file handle
    prfqpc_pszExe       dd  ?   ;Executable-file name
    prfqpc_pCategory    dd  ?   ;Program category
    prfqpc_pErrorId     dd  ?   ;Error identity
PRFQUERYPROGRAMCATEGORYPARMS ends
```

PRFQUERYPROGRAMHANDLEPARMS

API packet containing the parameters of a PrfQueryProgramHandle call.

```
PRFQUERYPROGRAMHANDLEPARMS struc
    prfqph_hini            dd   ?   ;Initialization-file handle
    prfqph_pszExe          dd   ?   ;Executable-file name
    prfqph_phprogArray     dd   ?   ;Array of program handles
    prfqph_cchBufferMax    dd   ?   ;Maximum length of array (in bytes)
    prfqph_pulCount        dd   ?   ;Number of program handles returned
    prfqph_pulLength       dd   ?   ;Length of returned data
    prfqph_pErrorId        dd   ?   ;Error identity
PRFQUERYPROGRAMHANDLEPARMS ends
```

PRFQUERYPROGRAMTITLESPARMS

API packet containing the parameters of a PrfQueryProgramTitles call.

```
PRFQUERYPROGRAMTITLESPARMS struc
    prfqpt_hini            dd   ?   ;Initialization-file handle
    prfqpt_hprogGroup      dd   ?   ;Handle of program or group
    prfqpt_pTitles         dd   ?   ;Program information buffer
    prfqpt_cchBufferMax    dd   ?   ;Buffer length
    prfqpt_pulCount        dd   ?   ;Number of structures returned
    prfqpt_pulLength       dd   ?   ;Length of returned data
    prfqpt_pErrorId        dd   ?   ;Error identity
PRFQUERYPROGRAMTITLESPARMS ends
```

PRFREMOVEPROGRAMPARMS

API packet containing the parameters of a PrfRemoveProgram call.

```
PRFREMOVEPROGRAMPARMS struc
    prfrp_hini             dd   ?   ;Initialization-file handle
    prfrp_hprog            dd   ?   ;Program handle
    prfrp_pfSuccess        dd   ?   ;Success indicator
    prfrp_pErrorId         dd   ?   ;Error identity
PRFREMOVEPROGRAMPARMS ends
```

PRFRESETPARMS

API packet containing the parameters of a PrfReset call.

```
PRFRESETPARMS struc
    prfr_hab               dd   ?   ;Anchor-block handle
    prfr_pProfile          dd   ?   ;Profile-names structure
    prfr_pfSuccess         dd   ?   ;Success indicator
    prfr_pErrorId          dd   ?   ;Error identity
PRFRESETPARMS ends
```

PRFWRITEPROFILEDATAPARMS

API packet containing the parameters of a PrfWriteProfileData call.

```
PRFWRITEPROFILEDATAPARMS struc
    prfwpd_hini            dd   ?   ;Initialization-file handle
    prfwpd_pszApp          dd   ?   ;Application name
    prfwpd_pszKey          dd   ?   ;Key name
    prfwpd_pData           dd   ?   ;Value data
    prfwpd_cchDataLen      dd   ?   ;Size of value data
    prfwpd_pfSuccess       dd   ?   ;Success indicator
    prfwpd_pErrorId        dd   ?   ;Error identity
PRFWRITEPROFILEDATAPARMS ends
```

PRFWRITEPROFILESTRINGPARMS

API packet containing the parameters of a PrfWriteProfileString call.

```
PRFWRITEPROFILESTRINGPARMS struc
    prfwps_hini          dd   ?   ;Initialization-file handle
    prfwps_pszApp        dd   ?   ;Application name
    prfwps_pszKey        dd   ?   ;Key name
    prfwps_pszData       dd   ?   ;Text string
    prfwps_pfSuccess   . dd   ?   ;Success indicator
    prfwps_pErrorId      dd   ?   ;Error identity
PRFWRITEPROFILESTRINGPARMS ends
```

--- END --

END OF DOCUMENT